VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Vizualizacie plánu vykonávaní dotazů

Visualization of Query Execution Plans

2017 Marek Horáček

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Marek Horáček**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Vizualizacie plánu vykonávaní dotazů
Visualization of Query Execution Plans

Jazyk vypracování: čeština

Zásady pro vypracování:

Plán vykonávání dotazů určuje přístupové cesty k jednotlivým tabulkám používaným dotazem a také pořadí jejich spojování. Cílem této práce je vytvořit webové rozhraní, které bude vizualizovat plány vykonávání dotazů.

Úkoly:
1. Nastudujte plány vykonávání dotazů v dnešních databázových systémech.
2. Naimplementujte webové rozhraní, které bude schopné:
a) vykonávat dotazy nad vytvořenou databází,
b) spouštět a vizualizovat plány vykonávání dotazů nad vytvořenou databází.
3. Vyhodnoťte možnosti dnešních databázových systémů z hlediska viyualiyace plánů vykonávání dotazů.

Seznam doporučené odborné literatury:

[1] Ray Rankins, Paul Bertucci, Chris Gallelli, Alex T. Silverstein. Microsoft SQL Server 2014 Unleashed. Sams Publishing; 1 edition, 2015.
[2] Edward Whalen and Mitchell Schroeter. Oracle Performance Tuning. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002.

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Peter Chovanec, Ph.D.**

Datum zadání: 01.09.2016

Datum odevzdání: 28.04.2017

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

VYSOKÁ ŠKOLA BÁŇSKÁ-TECHNICKÁ UNIVERZITA * Fakulta elektrotechniky a informatiky * 1D16 * OSTRAVA *

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 28.4.2017 .....

Na tomto místě bych rád poděkoval vedoucímu mé bakalářské práce panu Ing. Petrovi Chovancovi Ph.D. za odbornou pomoc a celé své rodině za každodenní podporu a motivaci.

Abstrakt

Tato bakalářská práce se zabývá možnostmi vizualizace plánů vykonání dotazů v dnešních databázových systémech. První část práce je zaměřena na obecné uvedení problematiky optimalizace informačních systémů a databázových aplikací, přičemž zdůrazňuje význam používání plánů vykonání dotazů při hledání problematických SQL dotazů. Druhá část práce podrobně popisuje plány vykonání dotazů v jednotlivých databázových systémech a uvádí možnosti, které tyto databázové systémy z hlediska plánů vykonání dotazů nabízejí. Na závěr jsou popsány a srovnány vizualizované plány vykonání dotazů napříč různými databázovými systémy.

Klíčová slova: databázový systém, SQL dotaz, plán vykonání dotazu, optimalizace

Abstract

This thesis deals with the possibilities of visualization query execution plans in database systems. The first part focuses on the general issue of optimizing database applications, highlighting the importance of using the query execution plans in the search for problematic SQL queries. The second part describes in detail the query execution plans in various database systems, and presents options of these database systems in terms of query execution plans. In conclusion are described and compared visualized query execution plans in current database systems.

Key Words: database system, SQL query, query execution plan, optimization

Obsah

Seznam použitých zkratek a symbolů

CPU	–	Central Processing Unit
GUID	–	Globally Unique Identifier
I/O	–	Input/Output
IOT	–	Index-Organized Table
SŘBD	–	Systém řízení báze dat
SQL	–	Structured Query Language

Seznam obrázků

Seznam tabulek

Seznam výpisů zdrojového kódu

1 Úvod

Drtivá většina aplikací a informačních systémů, které se dnes používají, potřebují nějakým způsobem trvale ukládat data. Nejrozšířenějším typem uložišť pro tyto aplikace jsou relační databázové systémy. Ve většině aplikací, které komunikují s databázemi, se dříve nebo později mohou vyskytnout výkonnostní problémy a je potřeba začít s optimalizací. Výkonnostně slabá místa aplikace se dají jen velmi těžko odhadnout před ostrým nasazením, jelikož výkonnostní problémy se typicky projeví až tehdy, kdy počet uživatelů nebo objem dat překročí náš původní předpoklad. Možností, jak postupovat při optimalizaci, je mnoho a dají se u nich používat různé nástroje a postupy. Pravidlem však je, že při optimalizaci by se mělo vždy začínat u samotné aplikace a postupovat směrem k databázovému serveru.

Jedním z kroků při optimalizaci informačních systémů je optimalizace SQL dotazů, pro kterou je jedním z nejmocnějších nástrojů plán vykonání dotazu. Plány vykonání dotazu nás mohou velmi podrobně informovat, jakým způsobem byly provedeny konkrétní SQL dotazy a je možné si z nich odvodit další možné kroky optimalizace. Plány vykonání dotazů jsme schopni zjistit ve většině databázových systémů (SŘBD) a to v různých způsobech zobrazení. Nejlepší variantou pro zobrazení způsobu provedení SQL dotazu je vizualizace plánu vykonání dotazu. Díky těmto vizualizovaným plánům se docílí velmi dobré čitelnosti a rychlejšího pochopení zpracování dotazů.

Cílem této bakalářské práce je uvést problematiku optimalizace informačních systémů, popsat plány vykonání dotazů v jednotlivých databázových systémech a následně uvést možnosti vizualizace těchto plánů. V rámci této práce byl vytvořen systém pro vizualizaci plánů vykonání dotazů, díky kterému je možné porovnat možnosti dnešních databázových systémů z hlediska vizualizace plánů vykonání dotazů.

2 Optimalizace výkonu

2.1 Obecně o optimalizaci

Již při vývoji aplikace by měl vývojář brát v úvahu výsledný výkon aplikace a snažit se o jeho maximalizaci. Většinou se po dokončení návrhu aplikace provádí simulace provozu a zkoumání, jak se aplikace chová při zatížení. Pokud jsou někde patrné výkonnostní problémy, pak se přistupuje k optimalizaci. Často se však optimalizace aplikace při návrhu podcení a pustí se do ostrého provozu, aniž by se výkonnostně otestovala. Poté se v mnoha scénářích stává, že aplikace není dostatečně rychlá a s přírůstkem počtu uživatelů a nárůstu objemu dat, začne vykazovat výkonnostní problémy. V takovém případě je třeba začít aplikaci optimalizovat, tedy snažit se o zvýšení jejího výkonu. Optimalizaci je možné provádět na různých vrstvách aplikace, od nichž se odvíjejí i prostředky a postupy, které při ní použijeme [2, 5]. Optimalizace aplikace se dá rozdělit do těchto skupin:

- Optimalizace na straně databázového serveru
  - tato optimalizace začíná výběrem vhodného hardwaru, posléze jeho konfigurací a případným rozložením zátěže jednotlivých komponent. Typicky se uvažuje o použití většího počtu diskových uložišť, díky čemuž dochází k rychlejšímu čtení a zápisu dat. Případně se mohou zavést tzv. serverové clustery [4], které představují spojení více databázových serverů, které se tváří jako jeden jediný, byť každý tento server disponuje vlastním hardwarem a zodpovídá za vlastní část databáze. Optimalizací databázového serveru se nebudu dále zabývat, protože to je nad rámec této bakalářské práce.
- Optimalizace na straně aplikace
  - velké výkonnostní problémy může přinést aplikace, která se dotazuje do databáze pro data, která nepotřebuje anebo se na ně dotazuje opakovaně, i když by si je mohla uchovávat už po prvním dotazu. V rámci optimalizace aplikace se používá tzv. Cachování a Connection Pooling [2]. Cachování představuje dočasné ukládání dat získaných z databáze v aplikaci, aby se zamezilo opakovanému dotazování na stejná data. Connection Poolingem se zabraňuje zbytečnému vytváření nových připojení na databázi, místo toho se používají již vytvořená, nepoužívaná připojení. Optimalizací na straně aplikace se nebudu dále zabývat, protože to je nad rámec této bakalářské práce.
- Optimalizace návrhu schématu databáze
- Optimalizace SQL dotazů

2.2 Optimalizace návrhu schématu databáze

Návrh schématu databáze je jedním z nejdůležitějších bodů vývoje informačních systémů a měl by mu být věnován dostatek času a ostatních prostředků. Samotný návrh se nedá příliš zobecnit, jelikož správný návrh schématu vždy souvisí s konkrétním scénářem použití. Většinou se musí uvažovat nad tím, které typy dotazů a ve kterých scénářích budou použity. Jako příklad se dá uvést přidání indexu do určité tabulky, čímž dojde ke zrychlení vyhledávání na úkor rychlosti vkládání. Ještě před samotným návrhem schématu bychom měli zvažovat, jaká potenciální data mohou být v databázi uložena, kolik uživatelů bude s databází pracovat a také s jakou frekvencí se budou na data dotazovat. Tomu musíme přizpůsobit celkový návrh. Při návrhu schématu je potřeba určit datové typy jednotlivých sloupců tabulek, vhodně zvolit primární klíče, zvolit datové struktury jednotlivých tabulek, vytvořit indexy a pokud to je nutné, přistoupit k normalizaci či denormalizaci schématu databáze [2].

2.2.1 Datové typy

Pro každý sloupec tabulky bychom měli volit co nejmenší datový typ, kterým je možné korektně uložit záznamy daného sloupce. Pokud bychom například chtěli uložit číslo a zvolili bychom datový typ varchar, zbytečně by každý záznam zabíral více místa na disku a v paměti. Dále u datových typů jako je varchar je také potřeba určit korektní rozsah, jelikož při zbytečně velkém rozsahu by opět záznamy zabíraly více místa na disku, než je nezbytně nutné. Dalším důležitým pravidlem pro definici sloupce tabulky je zamezení vkládání NULL hodnot, kde je to možné. Pokud například u sloupce nastavujeme výchozí hodnotu sloupce, nemá smysl, aby sloupec byl definován jako NULL. Se sloupci, u kterých je povolená hodnota NULL, se hůře pracuje, stěžují práci optimalizátoru dotazu a komplikují vytváření statistik. Opět je potřeba u nich alokovat více paměti, než je nezbytně nutné.

Velmi důležitý je také výběr datových typů primárních klíčů. Jak již bylo naznačeno výše, některé datové typy se hůře porovnávají než jiné, což hraje velkou roli například u spojování tabulek či pohledů. Jako vhodný primární klíč se dá považovat typ integer, případně pak složený klíč u spojovacích tabulek z vedlejších klíčů typu integer. Pokud bychom potřebovali ukládat unikátní identifikátor napříč různými databázemi, je vhodné používat typ GUID. U většiny SŘBD nějaký ekvivalent ke GUID existuje, může mít však jiný název.

2.2.2 Indexy

Velmi pozitivní vliv na výkon informačních systémů mohou mít indexy. Indexem se myslí struktura postavená vedle databázové tabulky, která zajišťuje rychlejší vyhledávání záznamů. V indexu jsou záznamy uspořádaný dle zvoleného klíče, resp. sloupce či sloupců tabulky. Pokud se rozhodneme používat tabulky bez indexů, pak při jakémkoli vyhledávání bude SŘBD nucen tabulku sekvenčně procházet. Naopak při použití indexu se počet procházených záznamů může razantně zmenšit. Přidání indexu do malé tabulky se nemusí výrazně projevit na době vykonání

dotazů, avšak u velkých tabulek jsou indexy téměř nepostradatelné a jejich používání razantně zkracuje dobu vykonání dotazů [5].

Pokud bychom se rozhodli nad tabulkou vytvořit index, musíme pečlivě zvážit, nad jakým sloupcem tento index vytvoříme, jelikož index se pro vyhledávání použije jen tehdy, je-li vyhledávacím predikátem právě indexovaný sloupec. Taktéž si musíme uvědomit, že vytvoření indexu sice zvýší rychlost vyhledávání, ale současně zpomalí ostatní operace, jako např. vkládání. Při vyhledávání záznamů přes index nemáme k dispozici rovnou vyhledávaná data, ale pouze ukazatel na jejich umístění v datové struktuře. Pokud bychom chtěli do indexu přidat kromě indexovaného sloupce ještě další sloupce, je možné při definici indexu zahrnout do indexu i další sloupce. Při vyhledávání v takovém indexu získáme vždy kromě indexovaného sloupce a ukazatele do datové struktury také navíc přidané sloupce. Ukazatel do datové struktury se v případě tabulky typu halda nazývá ROWID a v případě indexu, klíč indexu.

Typy indexů

- **B-strom** – nejčastěji používaná struktura indexu v databázích. Jedná se o strukturu balancovaného stromu, jehož cesty mezi kořenovým uzlem a všemi listy jsou stejné. Při vyhledávání se začíná v kořenovém uzlu a postupně se prochází stromová struktura směrem dolů s tím, že se vždy porovnává hledaná hodnota s klíčem v daném uzlu. Výhodou je také to, že všechny uzly ve stejném zanoření jsou propojeny, čímž lze procházet strom nejen do hloubky, ale také do šířky. Toho se využívá u rozsahového průchodu indexu [4].

- **Hash index** – struktura indexu je hašovací tabulka, do níž se záznamy vkládají dle hašovacího klíče vypočteného hašovací funkcí. Hašovací funkce rozděluje záznamy do jednotlivých kapes, ve kterých se nacházejí záznamy se stejným hašovacím klíčem. Data v indexu tedy nejsou uspořádána a není možné nad tímto indexem provádět rozsahový průchod [4].

- **Bitmap index** – tento typ indexu je vhodné použít u tabulek, které obsahují sloupce nabývající jen malé množství hodnot (v řádu jednotek), a často na základě těchto sloupců filtrujeme výsledek dotazu. Index obsahuje pro každý sloupec a hodnotu sloupce jeden bit, který určuje, zda záznam v daném sloupci obsahuje tuto hodnotu. Velkou nevýhodou je složitost aktualizace indexu. Je tedy vhodné jej použít jen u tabulek, ve kterých se záznamy nemění anebo mění jen velmi málo [4].

2.2.3 Fyzická implementace SŘBD

Veškerá data v databázi jsou uložena v předem definovaných strukturách [4]. Veškeré datové struktury jsou v uložišti rozděleny na tzv. bloky(stránky). Pomocí těchto bloků se dá ve struktuře orientovat a efektivně přistupovat na požadované místo v uložišti. Pokud bychom chtěli vytvořit v databázi tabulku, můžeme si vybrat, do jaké datové struktury má být uložena. Ve většině SŘBD se využívají hlavně dvě struktury [5]:

- **Tabulka typu halda** – nesetříděný seznam dat uložených na disku, kde každý záznam je označen unikátním identifikátorem (ROWID), ze kterého se dá zjistit, v jakém bloku se záznam nachází. Výhodou této struktury je velmi rychlé vkládání záznamů.

- **Index** – v kapitole 2.2.2 byly popisovány indexy jako pomocná struktura postavená vedle datové struktury (tabulky), jenž má zajistit rychlejší vyhledávání. Index však nemusí být jen pomocná struktura, ale může být i strukturou pro uložení dat samotných. Ve většině SŘBD se jako indexová struktura používá B-strom, který představuje setříděnou množinu záznamů podle určitého klíče (typicky primárního klíče tabulky). V listech tohoto stromu je uložen kompletní seznam dat pokrývající veškeré sloupce tabulky. V jednotlivých SŘBD se index jako datová struktura nazývá následovně:

  - Oracle – Index-organized table (IOT)
  - MS SQL Server – Clustered Index
  - PostgreSQL – tento SŘBD používá jako datovou strukturu jen tabulku typu halda.
  - MySQL – Clustered Index

2.2.4 Normalizace a denormalizace

Normalizace databázového schématu je soupis pravidel, kterými bychom se měli při návrhu schématu řídit, abychom získali schéma bez redundancí dat, schéma, nad kterým bude možné efektivně provádět SQL dotazy, a schéma, ve kterém nebude docházet k tzv. aktualizačním anomáliím [2]. Normalizace tedy řeší:

- Redundanci dat – pokud se data opakují ve více entitách(tabulkách), hovoříme o redundanci.

- Aktualizační anomálie – pokud schéma obsahuje redundantní data, pak při jejich modifikaci v jedné entitě dojde k nesouladu hodnot v entitě jiné, tím vznikají aktualizační anomálie.

Při normalizaci schématu se musejí zachovat všechny původní závislosti a také původní data. Míra normalizace schématu je rozdělena do několika normálních forem. Aby schéma bylo v některé normální formě, musí splňovat předem definovaná pravidla. Každá normální forma má svá vlastní pravidla, která je třeba dodržet. Většina tabulek by měla být alespoň ve 3. NF. Ostatní jsou spíše doplňkové a využívají se jen v určitých scénářích.

Opakem normalizace je denormalizace. Pokud se dostaneme do situace, kdy normalizované schéma pro nás není dostatečně výkonné a je potřeba výkon zvýšit, můžeme sáhnout po denormalizace. Tento proces je jakýmsi umělým vytvářením redundancí a závislostí uvnitř tabulek. Denormalizací se typicky myslí sjednocení více tabulek v jednu. Takovou akcí můžeme sice ušetřit operaci spojení, ale může nám to přinést jiné problémy a většinou se denormalizace používá až jako krajní řešení ve velmi specifickým případech.

2.3 Optimalizace SQL dotazů

SQL dotazy by se měly optimalizovat v celém životním cyklu aplikace, jelikož při narůstání objemu dat se může stát dříve plynulá část aplikace pomalou a tomu by se měly přizpůsobit i SQL dotazy, aby se dané části aplikace zvýšil výkon. Optimalizace SQL dotazů je relativně zdlouhavá záležitost a je tedy rozumné se vždy zaměřit na ty části aplikace, potažmo ty SQL dotazy, které jsou nejvíce problematické a aplikaci nejvíce zatěžují. Ke zjištění, jaké dotazy nám chodí na databázový server, můžeme využívat různé nástroje, typicky zabudované již v konkrétních administračních nástrojích pro dané SŘBD. Při psaní SQL dotazů bychom si měli uvědomit, že operace ORDER BY, HAVING a GROUP BY jsou náročnější na zpracování, než např. SELECT, WHERE nebo JOIN. Je tedy dobré se při psaní SQL dotazů zbavit nadbytečných třídění, klauzulí DISTINCT a jiných. Dále pokud píšeme složitější dotaz, ve kterém filtrujeme záznamy, měli bychom se snažit filtraci aplikovat co nejdříve, aby nedocházelo ke zpracování záznamů, které nakonec těsně před dokončením dotazu zahodíme. Hlavní pravidlo je však to, že bychom nikdy neměli dotazem vracet více dat, než potřebujeme. SQL dotaz by tedy neměl:

- Vracet více řádků, než je potřeba
  - typicky se jedná o případ, kdy aplikace není schopná zobrazit veškerá získaná data najednou, ale je potřeba data rozložit na více stránek, například se může jednat o diskuzní fórum, na kterém je zobrazeno třeba jen 50 příspěvků na každé stránce a uživatel má možnost mezi těmito stránkami přecházet. Toto bychom měli vyřešit tzv. stránkováním, tedy získáním jen nezbytného počtu řádků, které jsme schopni zobrazit a až při přechodu na další stránku získat řádky další.
- Vracet veškeré sloupce ze spojení více tabulek
  - pokud například chceme získat všechna data z jedné tabulky, kterou spojujeme z více tabulkami, dotaz by určitě neměl vypadat následovně:

    ```
    SELECT * FROM tab1
    JOIN tab2 ON tab1.id = tab2.tab1_id;
    ```

    Ale měl by vypadat například takto:

    ```
    SELECT tab1. * FROM tab1
    JOIN tab2 ON tab1.id tab2.tab1_id;
    ```
- Vracet veškeré sloupce
  - měli bychom se vždy vyhnout dotazům typu: `SELECT * FROM table` a to proto, že sice v době psaní dotazu můžeme skutečně chtít veškeré sloupce, ale zaprvé nám při psaní kódu nic nenapovídá, jaká data opravdu získáváme a zadruhé pokud bychom v budoucnu do tabulky nějaký sloupec přidali, pak bychom na všech takových typech dotazu získávali nežádoucí sloupec navíc, tedy museli bychom dotazy opět přepisovat.

Pokud jsme se ujistili, že dotaz vrací jen data, která skutečně chceme, ale stejně je pro nás tento dotaz příliš pomalý, měli bychom se podívat na vnitřní zpracování tohoto dotazu. Informaci o tom, jaký způsobem se dotaz provedl, nám poskytne plán vykonání dotazu. Nejprve je však potřeba si vysvětlit, jak probíhá zpracování SQL dotazu v databázovém serveru.

3 Zpracování SQL dotazu

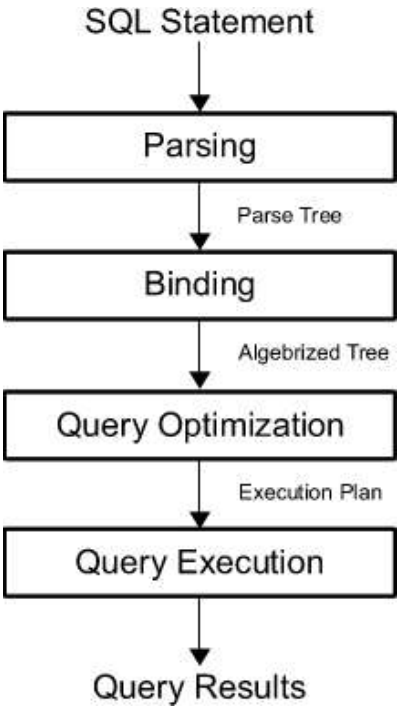
Při odeslání dotazu na databázový server se musí provést celá řada operací, než je dotaz vykonán a navrácen jeho výsledek. Na obrázku 1 je znázorněn proces zpracování SQL dotazu. Prvním krokem zpracování dotazu je nahlédnutí do cache paměti, zda v ní není uložen plán vykonání dotazu. Pokud v cache plán vykonání dotazu je, pak se rovnou provede, v opačném případě se zpracování předá Procesoru dotazu. Procesor dotazu v prvním kroku provede parsování SQL dotazu, čímž se procesor pokusí původní dotaz přepsat na výhodnější podobu dotazu. Následuje vytvoření tzv. Logického plánu dotazu, jenž vzniká nahrazením SQL syntaxe za sekvenci operací relační algebry, jejíž strukturou je většinou strom, jehož každý uzel představuje určitou logickou operaci. Logický plán dotazu se pošle Optimalizátoru dotazu, který má za úkol logický plán zpracovat, nahradit jednotlivé logické operace za fyzické operace a vyhledat nejlepší fyzický plán dotazu. Po získání fyzického plánu se tento plán použije pro získání dat z uložiště a následné navrácení výsledku klientovi [2].

I když se to tak nemusí zdát a psaní rozumných SQL příkazů je důležité, tak SQL jazyk je deklarativní, tedy při psaní SQL dotazu sdělujeme databázovému serveru informaci o tom, jaká data chceme, aby nám poskytl, nikoli způsob, jakým nám má tato data poskytnout.

3.1 Optimalizátor dotazu

Optimalizátor dotazu vyhodnocuje různé způsoby, kterými může být SQL dotaz proveden, a následně vybírá ten nejlepší způsob, neboli plán vykonání dotazu. Optimalizátor dotazu se rozhoduje mezi více plány vykonání dotazu na základě odhadované ceny jednotlivých plánů a vybírá plán s nejnižší cenou. Záleží na konkrétním SŘBD, zda optimalizátor prochází veškeré možné plány vykonání dotazu a vybírá skutečně ten nejlepší, anebo se spokojí s prvním uspokojivým plánem, tedy plánem, který ohodnotí jako efektivní a současně odhadne, že doba na nalezení rychlejšího plánu by byla delší, než doba pro samotné provedení dotazu s aktuálním plánem vykonání dotazu [3].

Hlavní součást práce optimalizátoru je generování kandidátních plánů vykonání dotazů, tedy plánů, které vracejí stejná data založená na požadavku z přijatého SQL dotazu. Takových plánů může být opravdu velké množství a záleží na konkrétní implementaci optimalizátoru jednotlivých SŘBD, zda se tyto plány vyhledávají opravdu všechny, anebo jen část. K tomu, aby optimalizátor byl schopen vybrat mezi kandidátními plány ten nejlevnější, ohodnocuje jednotlivé kroky daných plánů dle požadavků na různé zdroje, jako např. I/O, CPU nebo paměť. Veškeré části plánu vykonání dotazu jsou tedy ohodnoceny a odhaduje se tím jejich cena, která se počítá mimo jiné z množství vstupních dat jdoucích do konkrétní operace plánu. Bohužel, SŘBD nemá uloženy informace o počtu záznamů v jednotlivých tabulkách. Zjištění, kolik záznamů vstupuje do jednotlivých operací plánu vykonání dotazu, nám zajišťují statistiky. Je tedy zřejmé, že pokud se statistiky příliš neudržují a nejsou aktuální, pak může docházet k volbě neefektivních plánů vykonání dotazu a tedy k celkovému zpomalení zpracovávání dotazů.



Obrázek 1: Zpracování SQL dotazu

3.2 Statistiky

Jak již bylo naznačeno výše, statistiky jsou cenným pomocníkem optimalizátoru, jelikož mu poskytují informaci o odhadovaném množství řádků jdoucích do jednotlivých operací plánů vykonání dotazů, informace o použitelných indexech, atd. Statistik je několik typů:

- Tabulkové - obsahují počet záznamů v tabulce a jiné.
- Sloupcové - obsahují počet různých hodnot ve sloupci, počet NULL hodnot, histogramy a jiné.
- Indexové - obsahují počet listových stránek indexu a v případě stromové struktury indexu také výšku stromu.

Jelikož statistiky hrají významnou roli při výběru vhodného plánu vykonání dotazu pro provedení SQL dotazu, je velmi důležité, aby statistiky obsahovaly korektní a aktuální data, jinak by se mohl pro provedení dotazu volit neefektivní plán vykonání dotazu [3]. Statistiky by se měly tedy pravidelně udržovat a aktualizovat.

3.3 Hinty

Většinou optimalizátor dotazu provádí práci výběru plánů vykonání dotazů velmi dobře a není potřeba do tohoto výběru nijak zasahovat. Někdy se však může stát, že optimalizátor není scho-

pen odhadnout, že by bylo vhodné použít jiný plán, který by však byl lepší volbou. V takovém případě je vhodné použít umělého ovlivnění optimalizátoru k tomu, aby například využil jinou operaci, případně jiný přístup pro vyhledávání dat. Tento zásah do práce optimalizátoru dotazu nám umožňují hinty [3].

4 Plán vykonání dotazu

Plán vykonání dotazu představuje způsob provedení SQL dotazu. Způsobů, jak provést konkrétní SQL dotaz, může být mnoho a všechny spojuje jen to, že vracejí stejný výsledek. Plán vykonání dotazu si můžeme představit jako stromovou strukturu reprezentující cestu pro dosažení požadovaného výsledku definovaného SQL dotazem, přičemž každý uzel stromu představuje určitý krok plánu vykonání dotazu a zastupuje určitou fyzickou operaci [1].

4.1 Základní operace plánu vykonání dotazu

Operací plánu vykonání dotazu je celá řada a můžeme je rozdělit do následujících skupin:

- operace přístupu
- operace spojení
- ostatní operace

4.1.1 Operace přístupu

- **Sekvenční průchod tabulkou** – dochází k průchodu všech záznamů tabulky a postupně se jednotlivé záznamy načítají do výsledku. Pokud je výběr z tabulky omezen nějakou podmínkou, dochází navíc k filtraci jednotlivých záznamů a do výsledku se načítají pouze ty, které podmínku splňují.
- **Přístup do tabulky pomocí ROWID** – dochází k načtení konkrétního bloku tabulky, v němž je uložen záznam s vyhledávaným ROWID.
- **Bodový průchod indexem** – v případě indexu typu B-strom dochází k průchodu stromu od kořene přes jednotlivé uzly do listového uzlu, ve kterém se nachází záznam pro vyhledávaný klíč indexu.
- **Rozsahový průchod indexem** – v tomto případě dochází ke stejnému průchodu struktury indexu jako při bodovém průchodu, avšak dojde se do prvního listu, jehož klíč splňuje vyhledávací podmínku, a následuje průchod mezi dalšími listy stromu, které podmínku splňují.
- **Úplný průchod indexu** – v tomto případě se procházejí veškeré listové uzly indexu typu B-strom. Typicky se tato operace použije v případě, že je pro nás vhodné získat uspořádanou množinu záznamů, kterou index nabízí.

4.1.2 Operace spojení

Vnořené cykly (Nested loops)

V tomto algoritmu se určí jedna relace za vnější a druhá relace za vnitřní zdroj, přičemž pro každý záznam vnějšího zdroje se ve vnitřním zdroji vyhledávají záznamy, které splňují podmínku spojení. Jedná se o nejprimitivnější způsob spojení a většinou se používá u spojování tabulek s malým množstvím záznamů. V ostatních případech se většinou vyplatí využít jiné operace spojení.

Hašované spojení (Hash join)

Tento typ spojení je vhodný u relací s velkým počtem záznamů. A je velmi vhodné, aby jedna ze spojovacích relací byla výrazně menší, než druhá. Hašovací algoritmus má dvě fáze. V první fázi se vybere menší relace, která se sekvenčně projde a každý spojovaný záznam z relace je uložen do příslušné hašovací tabulky spolu s hašovacím klíčem vypočteným hašovací funkcí. Hašovací funkce jsou různé, ale záměr těchto funkcí je rovnoměrně rozložit relaci do dílčích kapes, jejíchž záznamy budou mít stejnou hodnotu klíče vypočteného hašovací funkcí. Tyto kapsy jsou poté načteny do paměti a ve druhém fázi se sekvenčně procházejí záznamy větší relace a počítá se pro ně opět klíč hašovací funkcí a následně se nemusí pro každý záznam z větší relace projít sekvenčně první relace, ale stačí, když se projde pouze konkrétní kapsa pro daný hašovací klíč.

Spojení sléváním (Merge join)

Pro tento typ spojení je potřeba mít na vstupu setříděné relace podle klíče v podmínce spojení. Následuje porovnávání pro každý záznam z první relace se záznamy v druhé relaci s tím, že pokud právě porovnávaný záznam v druhé relaci je větší než aktuálně vyhledávaný záznam z první relace, pak se přeskočí na další záznam první relace, jelikož vyhledávaný záznam ve druhé relaci není.

4.1.3 Ostatní operace

Existují i další fyzické operace jako např. třídění, grupování nebo různé množinové operace.

4.2 MS SQL Server

SQL Server [1] rozlišuje zobrazení dvou typů plánu vykonání dotazu a sice odhadovaný a aktuální plán vykonání dotazu. Odhadovaný plán vykonání dotazu je vytvořen bez potřeby spouštět konkrétní dotaz a je vhodné jej použít tehdy, pokud získání aktuálního plánu vykonání dotazu trvá příliš dlouho, tedy v komplikovaných dotazech vracejících velké množství záznamů. Aktuální plán vykonání dotazu představuje právě použitý plán a je pro jeho získání potřeba dotaz provést. Oba typy plánů mohou být ve spoustě případech totožné, avšak začnou se poměrně rozcházet v případě, kdy má optimalizátor dotazu k dispozici zastaralé statistiky.

Tabulka 1: Popis atributů v MS SQL Serveru [10]

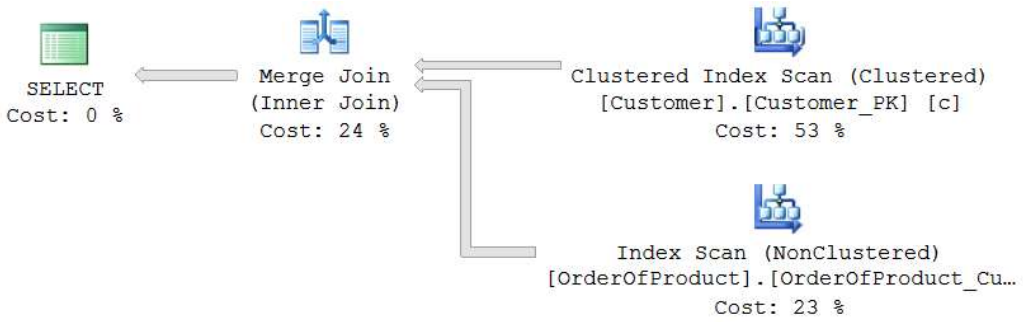
Atribut	**Popis**
StmtText	Pro řádky, které nejsou typu `PLAN_ROW`, obsahuje tento atribut text SQL dotazu. Pro řádky typu `PLAN_ROW` obsahuje atribut popis operace.
StmtId	Identifikátor SQL dotazu v rámci dávky dotazů.
NodeId	Identifikátor uzlu v aktuálním dotazu.
Parent	Identifikátor rodičovského uzlu.
PhysicalOp	Název fyzické operace. Jen u řádků typu `PLAN_ROW.`
LogicalOp	Název logické operace. Jen u řádků typu `PLAN_ROW`
Argument	Udává dodatečnou informaci o provedené fyzické operaci.
DefinedValues	Obsahuje seznam hodnot zavedených operátorem, které využívá procesor dotazu pro vykonání dotazu. Jen u řádků typu `PLAN_ROW.`
EstimateRows	Předpokládaný počet řádků na výstupu operátoru. Je u řádků typu `PLAN_ROW`.
EstimateIO	Překládaná cena I/O. Jen u řádků typu `PLAN_ROW`.
EstimateCPU	Překládaná cena CPU. Jen u řádků typu `PLAN_ROW`.
AvgRowSize	Předpokládaná průměrná velikost řádku (v bajtech).
TotalSubtreeCost	Předpokládaná cena operace a všech potomků.
OutputList	Obsahuje seznam sloupců vystupujících z dané operace.
Warnings	Obsahuje seznam varování spojených s aktuální operací.
Type	Typ operátoru. Pro hlavní uzel dotazu je zde uveden název T-SQL příkazu. Pro odvozené uzly je uvedeno `PLAN_ROW.`
Parallel	Příznak identifikující paralelní spuštění operátoru.
EstimateExecutions	Odhadovaný počet spuštění operátoru v rámci provedení aktuálního dotazu.

4.2.1 Formáty

SQL Server nabízí zobrazení plánů vykonání dotazu v různých formátech. Máme na výběr mezi grafickým zobrazením, zobrazením v textové podobě a XML formátu. Všechny tři způsoby zobrazení nabízejí stejné informace o plánu vykonání dotazu. Tabulka 1 shrnuje a popisuje atributy jednotlivých operátorů plánu vykonání dotazu.

Grafické zobrazení

Zobrazení plánu vykonání dotazu v grafické podobě je nejběžnějším způsobem zobrazení. Výhodou je hlavně rychlé a jednoduché čtení. Je možné zobrazit jak odhadované, tak aktuální plány vykonání dotazu. Podrobné informace o plánu vykonání dotazu je možné najít v ToolTipu, který má každý uzel plánu. Grafické zobrazení je dostupné v prostředí SQL Server Management Studio a jeho ukázka je na obrázku 2. Získání grafického plánu vykonání dotazu je možné z hlavní nabídky v záložce Query > Include Actual Execution Plan.



Obrázek 2: Ukázka plánu vykonání dotazu v SQL Server Management Studio

Textové zobrazení

O něco hůře čitelný způsob zobrazení je zobrazení v textové podobě. Výhodou tohoto formátu je přenositelnost, jelikož jsme schopni tento formát exportovat a uložit v textové podobě pro další použití.
Tento formát má 3 různé varianty:

- `SHOWPLAN_ALL` – touto variantou získáme podrobný popis odhadovaného plánu vykonání dotazu ve formě výpisu do SQL výstupu. Příkaz: `SET SHOWPLAN_ALL ON`.
- `SHOWPLAN_TEXT` – tato varianta nabízí jen omezené informace a ve formátu, který není příliš strukturovaný. Příkaz: `SET SHOWPLAN_TEXT ON`.
- `STATISTICS PROFILE` – stejný výstup jako u `SHOWPLAN_ALL`, jen se nejedná o odhadovaný, ale o aktuální plán vykonání dotazu. Příkaz: `SET STATISTICS PROFILE ON`.

XML formát

Tento způsob nabízí všechny podrobné informace o plánu vykonání dotazu. Formát XML je obecně velmi používaným formátem pro reprezentaci dat a je tedy dobrou volbou, pokud chceme data dále zpracovávat. Čitelnost takového plánu vykonání dotazu je poněkud složitější oproti grafickému zobrazení.
Tento formát má 2 varianty:

- `SHOWPLAN_XML` – zobrazení odhadovaného plánu vykonání dotazu. Příkaz pro jeho získání: `SET SHOWPLAN_XML ON`.
- `STATISTICS_XML` – zobrazení aktuálního plánu vykonání dotazu. Příkaz pro jeho získání: `SET STATISTICS_XML ON`.

4.2.2 Operátory plánu vykonání dotazu

- **Aggregate** – operace představuje některý z typů počítání výrazu. Jedná se o operace MIN, MAX, SUM, COUNT a AVG.
- **Assert** – operace má za úkol zjišťování splnění některé podmínky. Typickým příkladem může být ověřování, zda nějaká hodnota splňuje integritního omezení sloupce.
- **Table Scan** – sekvenční průchod tabulkou (viz. kapitola 4.1.1).
- **Clustered Index Scan** – úplný průchod klastrovaného indexu (viz. kapitola 4.1.1).
- **Clustered Index Seek** – bodový nebo rozsahový průchod klastrovaným indexem (viz. kapitola 4.1.1).
- **NonClustered Index Seek** – bodový nebo rozsahový průchod neklastrovaným indexem (viz. kapitola 4.1.1).
- **Key Lookup** – používá se pro vyhledání záznamu z klastrovaného indexu na základě klíče indexu.
- **RID Lookup** – používá se pro vyhledávání záznamu z tabulky na základě ukazatele na řádek tabulky (RID).
- **Nested Loops** – spojení vnořenými cykly (viz. kapitola 4.1.2).
- **Hash Match** – hašované spojení (viz. kapitola 4.1.2).
- **Merge Join** – spojení sléváním (viz. kapitola 4.1.2).
- **Compute Scalar** – tato operace reprezentuje výpočet určité hodnoty, např. pro podmínku filtrace či spojení.
- **Concatenation** – tento operátor přelévá hodnoty z více vstupů na jeden výstup. Typicky se jedná o operaci sjednocení (UNION ALL).
- **Constant Scan** – tento operátor má za úkol vkládat jeden nebo více konstantních řádků na výstup.
- **Distinct** – operátor odstraňuje duplicitní hodnoty ze vstupu.
- **Filter** – tento operátor čte záznamy ze vstupu a vrací pouze ty hodnoty, které vyhověly podmínce pro průchod.
- **Sort** – operátor pro setřídění všech vstupních řádků. Je možné provést vzestupné i sestupné třídění.
- **Stream Aggregate** – operace grupování setříděných záznamů.

- **Hash Match Aggregate** – operace využívající dočasnou hašovací tabulku pro grupování záznamů. Výhodou je, že nevyžaduje setříděné vstupní záznamy.

4.3 Oracle

4.3.1 Formáty

Oracle [8] nabízí jen dva způsoby zobrazení plánů vykonání dotazů. Máme na výběr mezi grafickým zobrazením a tabulkou. Tabulka 2 shrnuje a popisuje ty nejdůležitější atributy jednotlivých operátorů plánu vykonání dotazu.

Grafické zobrazení

Grafické zobrazení je opět dobře čitelné a je asi nejběžněji používané. Grafické zobrazení je dostupné v prostředí Oracle SQL Developer a jeho ukázka je na obrázku 3. Získání grafického plánu vykonání dotazu je možné v okně SQL dotazu stiskem tlačítka Explain Plan.

OPERATION	OBJECT_NAME	CARDINALITY	COST
SELECT STATEMENT		11000	44
HASH JOIN		11000	44
Access Predicates			
OP.CUSTOMERID=C.CUSTOMERID			
INDEX	ORDEROFPRODUCT_CUSTOMERID_IDX	11000	10
TABLE ACCESS	CUSTOMER	5000	34
Other XML			

Obrázek 3: Ukázka plánu vykonání dotazu v Oracle SQL Developer

Uložení do tabulky

Oracle má pro plány vykonání dotazů připravenou tabulku `Plan_table`, do které při každém zavolání příkazu `EXPLAIN PLAN FOR SELECT * FROM [tabulka]` uloží výsledný plán vykonání dotazu. Z této tabulky se poté veškeré informace mohou získat s filtrací přes ID plánu, což je unikátní identifikátor daného plánu vykonání dotazu v tabulce. Cílová tabulka plánu vykonání dotazu lze měnit na jinou tabulku vytvořenou uživatelem. Získání plánu vykonání dotazu z tabulky lze příkazem `SELECT * FROM Plan_table`.

4.3.2 Operátory plánu vykonání dotazu

Zde je potřeba zdůraznit, že Oracle v plánech vykonání dotazů nerozlišuje průchod indexem a čtení z IOT. Pokud je některý operátor vázaný na index, může být rovněž použit pro čtení z IOT.

- **Full Table Scan** – sekvenční průchod tabulkou (viz. kapitola 4.1.1).
- **ROWID Scan** – přístup do tabulky pomocí ROWID (viz. kapitola 4.1.1).

Tabulka 2: Popis atributů v Oracle [7]

Atribut	**Popis**
STATEMENT_ID	Nepovinná hodnota identifikující SQL dotaz v rámci dávky dotazů.
PLAN_ID	Unikátní identifikátor plánu vykonání dotazu.
TIMESTAMP	Datum a čas generování plánu vykonání dotazu.
OPERATION	Název operace provedené v daném kroku. U prvního kroku plánu je zde uveden název SQL příkazu.
OPTIONS	Specifikující popis provedené operace.
OBJECT_OWNER	Jméno uživatele vlastnícího schéma, které obsahuje použitou tabulku nebo index.
OBJECT_NAME	Název tabulky nebo indexu.
OBJECT_ALIAS	Unikátní alias pro tabulku nebo pohled v SQL příkazu.
OBJECT_INSTANCE	Číslo korespondující s pozicí objektu v původním SQL příkazu.
OBJECT_TYPE	Popisná informace daného objektu.
OPTIMIZER	Aktuální mód optimalizátoru.
ID	Identifikátor kroku plánu vykonání dotazu.
PARENT_ID	Identifikátor nadřazeného kroku plánu vykonání dotazu.
DEPTH	Úroveň zanoření ve struktuře plánu vykonání dotazu.
COST	Odhadovaná cena kroku plánu vykonání dotazu a jeho potomků.
CARDINALITY	Odhadovaný počet řádků vstupujících do dané operace.
BYTES	Odhadovaný počet bajtů dat vstupujících do dané operace.
OTHER_TAG	Dodatečná informace blíže popisující význam sloupce OTHER.
OTHER	Ostatní informace o daném kroku plánu vykonání dotazu.
CPU_COST	Cena operace z hlediska zátěže CPU odhadovaná optimalizátorem dotazu.
IO_COST	Cena I/O operace odhadovaná optimalizátorem dotazu.
ACCESS_PREDICATES	Podmínka pro získání řádků z datové struktury. Patří zde např. klíč spojení relací.
FILTER_PREDICATES	Podmínka pro filtraci řádků před jejich použitím.
PROJECTION	Seznam sloupců a jiných hodnot vystupujících z dané operace.

- **Index Unique Scan** – bodový průchod indexem (viz. kapitola 4.1.1) s tím, že vrací pouze jeden záznam. Používá se tedy pro získání unikátního záznamu.
- **Index Range Scan** – rozsahový průchod indexem (viz. kapitola 4.1.1).
- **Index Full Scan** – úplný průchod indexem (viz. kapitola 4.1.1).
- **Fast Full Index Scan** – úplný průchod indexem (viz. kapitola 4.1.1). Rozdíl oproti Index Full Scanu je v tom, že po této operaci nenásleduje dohledávání záznamů v tabulce. Veškeré potřebné informace jsou tedy obsaženy v indexu.
- **Bitmap Index Single Value** – použití bitmap indexu pro získání jedné hodnoty.
- **Bitmap Index Range Scan** – rozsahový průchod bitmap indexem.
- **Bitmap Index Full Scan** – úplný průchod bitmap indexem.
- **Nested Loops** – spojení vnořenými cykly (viz. kapitola 4.1.2).
- **Hash Join** – hašované spojení (viz. kapitola 4.1.2).
- **Merge Join** – spojení sléváním (viz. kapitola 4.1.2).
- **Sort** – operace pro setřídění záznamů. Existuje několik variant:
  - **Unique** – tříděním zajišťuje eliminaci duplicitních záznamů.
  - **Join** – třídění využívané pro operaci Merge-join.
  - **Order By** – třídění založené na klauzuli ORDER BY.
  - **Group By** – třídění založené na klauzuli GROUP BY.
  - **Aggregate** – vrací jeden řádek jako výsledek grupování více řádků.
- **Count** – operace vracející počet vstupujících záznamů.
- **Filter** – operace, která na základě podmínky filtruje vstupní záznamy.

4.4 MySQL

4.4.1 Formáty

MySQL [2] nabízí zobrazení plánů vykonání dotazů v grafickém režimu, ve formátu JSON a v textovém formátu. Podrobnosti o plánu vykonání dotazu se v jednotlivých způsobech zobrazení liší. Tabulka 3 shrnuje a popisuje atributy jednotlivých operátorů plánu vykonání dotazu [9].

Tabulka 3: Popis atributů v MySQL

Atribut	Popis
`Id`	Číslo identifikující každý poddotaz v rámci SQL dotazu.
`Select_type`	Typ dotazu.
`Table`	Název tabulky nebo pohledu, nad kterým se provádí daná operace.
`Partitions`	Pokud se operace provádí nad rozdělenou tabulkou, pak se v tomto sloupci objeví ty části tabulky, ze kterých se data získala.
`Possible_keys`	Názvy použitelných indexů pro provedení dané operace.
`Key`	Název klíče (indexu), který se optimalizátor rozhodl použít.
`Key_len`	Délka použitého klíče (indexu).
`Ref`	Názvy sloupců nebo konstant použité pro vyhledávání v indexu či spojení relace.
`Rows`	Odhadovaný počet řádků vystupujících z dané operace.
`Filtered`	Odhadovaný počet procent řádků, které splní podmínku filtrace.
`Extra`	Dodatečná informace o provedení dotazu.

Grafické zobrazení

Grafické zobrazení nabízí jen omezený počet informací o plánu vykonání dotazu, avšak je vhledem ke snadné čitelnosti nejpoužívanější způsob zobrazení. Grafické zobrazení je dostupné v prostředí MySQL Workbench a jeho ukázka je na obrázku 4. Získání grafického plánu vykonání dotazu je možné po spuštění libovolného dotazu v záložce Execution Plan.

Textové zobrazení

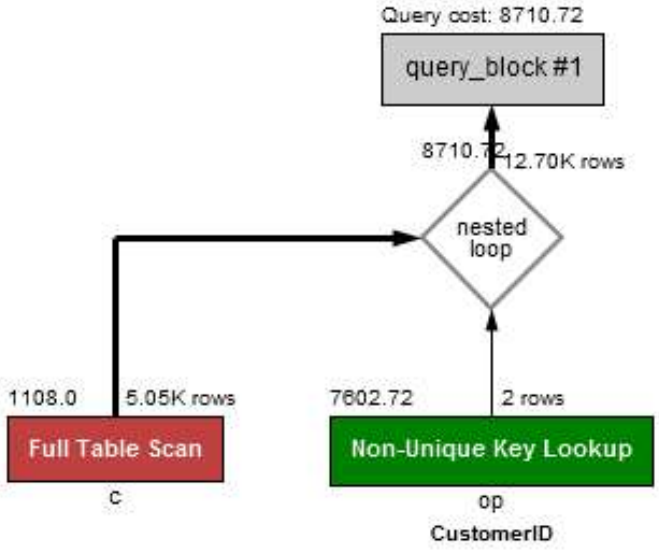
V tomto případě dojde k zobrazení plánu vykonání dotazu na SQL výstup, podobně jako u MS SQL Serveru. Pro získání plánu vykonání dotazu v tomto formátu je potřeba provést příkaz `EXPLAIN SELECT * FROM [tabulka]`. Tento typ zobrazení poskytuje omezené informace o plánu vykonání dotazu.

Formát JSON

Tento typ zobrazení poskytuje úplné informace o provedení plánu vykonání dotazu. Pro získání plánu vykonání dotazu slouží příkaz `EXPLAIN FORMAT = JSON SELECT * FROM [tabulka]`.

4.4.2 Operátory plánu vykonání dotazu

MySQL má poněkud specifický způsob popisu uzlů plánu vykonání dotazu. Nepopisuje jednotlivé uzly konkrétními fyzickými a logickými operacemi, ale jen klíčovými slovy, které konkrétní operace zastupují [9]. Při grafickém zobrazení plánu vykonání dotazu v prostředí Workbench



Obrázek 4: Ukázka plánu vykonání dotazu v MySQL Workbench

se tato klíčová slova nahrazují za konkrétní operace. V následujícím seznamu je u každého klíčového slova v závorce uvedena operace, která klíčové slovo v prostředí Workbench zastupuje. Důležité je také zmínit, že v MySQL se nerozlišuje průchod klastrovaného indexu a tabulky typu halda. Tedy např. operátor ALL znamená jak sekvenční průchod tabulkou, tak úplný průchod indexem.

- **System** (Single Row System Constant) – pro přístup do tabulky, která má pouze jeden řádek.
- **Const** (Single Row Constant) – sekvenční průchod tabulkou (viz. kapitola 4.1.1) s tím, že vrací pouze jeden záznam. Používá se při vyhledávání dle primárního klíče nebo podle unikátní hodnoty.
- **ALL** (Full Table Scan) – sekvenční průchod tabulkou (viz. kapitola 4.1.1).
- **Index** (Full Index Scan) – úplný průchod indexem (viz. kapitola 4.1.1).
- **Range** (Index Range Scan) – rozsahový průchod indexem (viz. kapitola 4.1.1).
- **Ref** (Non-Unique Key Lookup) – rozsahový průchod indexem (viz. kapitola 4.1.1) s použitím neunikátního indexu, tedy indexu, ve kterém se nacházejí záznamy se stejným klíčem. Tyto záznamy se sekvenčně čtou a přidávají k výsledku.

- **Eq ref** (Unique Key Lookup) – bodový průchod indexem (viz. kapitola 4.1.1).
- **Using filesort** – indikuje použití třídění.

4.5 PostgreSQL

Tabulka 4: Popis atributů v PostgreSQL

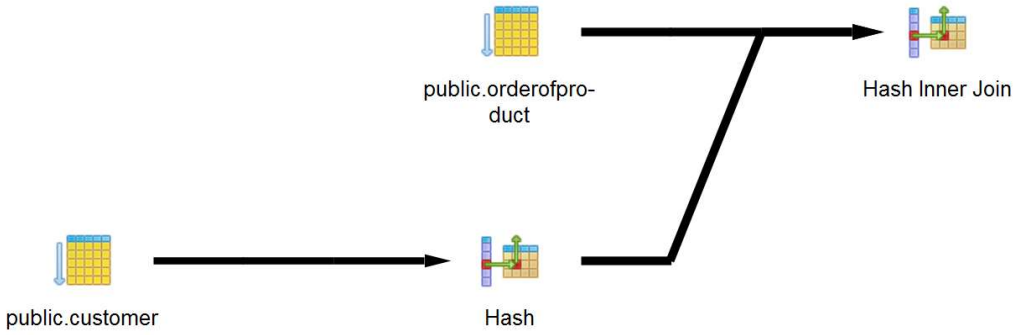
Atribut	Popis
`Startup-Cost`	Odhadovaná cena všech operací předcházejících tuto operaci.
`Total-Cost`	Součet cen vykonání všech předešlých operací a aktuální operace.
`Plan-Rows`	Odhadovaný počet řádků vystupujících z dané operace.
`Plan-Width`	Průměrná velikost řádků (v bajtech) vycházejících z dané operace.
`Node-Type`	Typ uzlu plánu vykonání dotazu.
`Join-Type`	Typ spojení.
`Hash-Cond`	Podmínka pro provedení hašovaného spojení.
`Merge-Cond`	Podmínka pro provedení spojení sléváním.
`Relation-Name`	Název tabulky nebo pohledu nad kterým se provádí daná operace.
`Alias`	Alias pro konkrétní tabulku nebo pohled.
`Filter`	Popis filtrace na základě podmínky uvedení v klauzuli WHERE.
`Subplan-Name`	Název vnořeného dotazu.
`Output`	Seznam sloupců získávaných danou operací.

4.5.1 Formáty

PostgreSQL [6] nabízí zobrazení plánů vykonání dotazů v grafickém režimu, v textovém režimu a v dalších formátech jako jsou XML, JSON a YAML. Formáty XML, JSON a YAML obsahují podrobné informace o plánu vykonání dotazu. Textové a grafické zobrazení obsahují informací o plánu vykonání dotazu méně. Tabulka 4 shrnuje a popisuje atributy jednotlivých operátorů plánu vykonání dotazu.

Grafické zobrazení

Grafické zobrazení je opět nejpřehlednější avšak představuje méně podrobné informace než např. formát XML. Grafické zobrazení je dostupné v prostředí pgAdmin a jeho ukázka je na obrázku 5. Získání grafického plánu vykonání dotazu je možné v okně SQL dotazu tlačítkem Explain.



Obrázek 5: Ukázka plánu vykonání dotazu v pgAdmin

Textové zobrazení

Jedná se o nestrukturovaný výpis na SQL výstup, ve kterém se poměrně špatně orientuje. Tento formát navíc není vhodný pro další zpracování a přenos plánu vykonání dotazu. Jedná se o výchozí nastavení zobrazení a je možné jej získat příkazem `EXPLAIN SELECT * FROM [tabulka]`.

Formáty XML, JSON a YAML

Tyto formát nabízejí stejně podrobné informace a jsou stejně čitelné. Příkaz pro získání v tomto formátu je `EXPLAIN (FORMAT [formát]) SELECT * FROM [tabulka]`.

4.5.2 Operátory plánu vykonání dotazu

- **Seq Scan** – sekvenční průchod tabulkou (viz. kapitola 4.1.1).
- **Index Scan** – rozsahový průchod indexem (viz. kapitola 4.1.1) následovaný přístupem do tabulky pomocí ROWID (viz. kapitola 4.1.1). V případě hash indexu se projde kapsa příslušná vypočtenému hašovacímu klíči a následně se získají data z tabulky pomocí ROWID.
- **Index Only Scan** – rozsahový průchod indexem (viz. kapitola 4.1.1) bez nutnosti následného průchodu tabulkou, jelikož všechna potřebná data byla získána z indexu.
- **Nested Loops** – spojení vnořenými cykly (viz. kapitola 4.1.2).
- **Hash Join** – hašované spojení (viz. kapitola 4.1.2).
- **Merge Join** – spojení sléváním (viz. kapitola 4.1.2).
- **Sort / Sort Key** – setřídění záznamů podle určitého klíče.
- **GroupAggregate** – operace grupování setříděných záznamů.
- **HashAggregate** – operace využívající dočasnou hašovací tabulku pro grupování záznamů. Výhodou je, že nevyžaduje setříděné vstupní záznamy.

- **Bitmap Heap/Index Scan** – na základě ROWID získaných z indexu se vytvoří bitmapa, ve které každý jeden bit značí blok tabulky, a následně se prohledávají bloky tabulky podle bitmapy. Operace je použita při získávání velkého počtu ROWID z indexu.

5 QEP - Nástroj pro vizualizaci plánů vykonání dotazů

Jedná se o systém umožňující spouštění SQL dotazů zadanými uživatelem, výpis výsledků dotazů a vizualizaci plánů vykonání dotazů. Systém byl vyvinut jako webová aplikace v jazyce C# nad platformou ASP.NET. Aplikace komunikuje s databázovými servery MS SQL Server, Oracle, PostgreSQL a MySQL. Tyto SŘBD byly vybrány jako nejrozšířenější zástupci a v případě potřeby by mohla být aplikace rozšířena i o další. V těchto SŘBD může být vytvořena libovolná databáze a nad touto databází je možné z aplikace spouštět SQL dotazy. Pro účely této bakalářské práce jsem vytvořil shodné databáze ve všech SŘBD. Schéma této databáze je popsáno v kapitole 5.2. Aplikace nabízí i možnost vytvoření a naplnění databáze, avšak to nebylo přímo v zadání bakalářské práce, a proto je tato možnost v aplikaci zatím znepřístupněna. Aplikace je dostupná na `http://db.cs.vsb.cz/qep/`.

5.1 Popis aplikace

5.1.1 Popis funkcionality

Aplikace obsahuje stránky QEP a Datový model. Stránka QEP obsahuje následujících 6 záložek:

- SQL – slouží pro vložení SQL dotazu, výběr cílových SŘBD a spuštění SQL dotazu.
- MS SQL Server – slouží k zobrazení výstupu SQL dotazu provedeném v MS SQL Serveru. Obsahuje také informaci o celkovém počtu získaných záznamů a době provedení dotazu.
- Oracle – slouží k zobrazení výstupu SQL dotazu provedeném v Oracle. Obsahuje také informaci o celkovém počtu získaných záznamů a době provedení dotazu.
- MySQL – slouží k zobrazení výstupu SQL dotazu provedeném v MySQL. Obsahuje také informaci o celkovém počtu získaných záznamů a době provedení dotazu.
- PostgreSQL – slouží k zobrazení výstupu SQL dotazu provedeném v PostgreSQL. Obsahuje také informaci o celkovém počtu získaných záznamů a době provedení dotazu.
- Plán vykonání dotazu – slouží ke grafickému zobrazení plánu vykonání dotazu.

Při provádění SQL dotazu má uživatel možnost zvolit libovolný počet cílových SŘBD, avšak minimálně jeden. Po vykonání dotazu je uživateli povolen přístup do záložek jednotlivých SŘBD, který byl před vykonáním odepřen. Pokud uživatel chce zobrazit plány vykonání dotazu, má možnost v záložce SQL kliknout na tlačítko Zobrazit plán vykonání dotazu, čímž mu bude povolen přístup do záložky Plán vykonání dotazu, ve kterém se zobrazí plán vykonání dotazu pro každý zvolený SŘBD.

Stránka Datový model obsahuje popis testovacího datového modelu, jeho schéma a datový slovník.

5.1.2 Popis tříd a rozhraní

```
interface IDatabaseProxy<T>
{
   void CreateSchema(string query);
   void DeleteSchema(string query);
   int InsertData(string query);
   void InsertDataBulk(string query);
   DataTable Select(string query, out bool failed);
   List<T> GetQueryExecutionPlan(string query, out bool failed);
}
```

Výpis 1: Rozhraní IDatabaseProxy

Rozhraní `IDatabaseProxy` (viz. výpis 1) je rozhraní generického typu, které následně implementují třídy pracující s jednotlivými databázemi.

- `void CreateSchema(string query)` – funkce sloužící k vytvoření schéma databáze.
- `void DeleteSchema(string query)` – funkce sloužící k smazání schéma databáze.
- `int InsertData(string query)` – funkce sloužící k importu záznamů do databáze.
- `void InsertDataBulk(string query)` – funkce sloužící k importu dat do databáze za použití bulk operace.
- `DataTable Select(string query)` – funkce vrací výsledek dotazu ve formě Data Table.
- `List<T> GetQueryExecutionPlan(string query)` – funkce pro získání plánu vykonání dotazu pro SQL dotaz. Vrací seznam objektů reprezentujících uzly plánu vykonání dotazu v konkrétních SŘBD.

Seznam tříd implementující toto rozhraní:

- `MSSqlDbProxy` – třída pro komunikaci s MS SQL databází
- `OracleDbProxy` – třída pro komunikaci s Oracle databází
- `MySqlDbProxy` – třída pro komunikaci s MySQL databází
- `PostgreSqlDbProxy` – třída pro komunikaci s PostgreSQL databází

Třídy reprezentující uzly plánu vykonání dotazu jednotlivých SŘBD:

- `MsSqlPlanNode`
- `OraclePlanNode`

- `MySqlPlanNode`
- `PostgreSqlPlanNode`

Rozhraní `ITreeCreator` (viz. výpis 2) je rozhraní generického typu, které následně implementují třídy generující stromovou strukturu z objektů `MsSqlPlanNode`, `OraclePlanNode`, `MySqlPlanNode` nebo `PostgreSqlPlanNode`.

- `List<TreeNode> GetTreeNodes()` – funkce pro získání stromové stuktury plánu vykonání dotazu. Vrací seznam uzlů stromu.
- `void GetChilds(T parent, TreeNode treenode)` – funkce pro rekurzivní průchod potomků daného uzlu.
- `string GetToolTip(T node)` – funkce pro získání tooltipu daného uzlu.
- `string GetImageUrl(string operation)` – funkce pro získání cesty k obrázku pro konkrétní operaci plánu vykonání dotazu.

```
interface ITreeCreator<T>
{
   List<TreeNode> GetTreeNodes();
   void GetChilds(T parent, TreeNode treeNode);
   string GetToolTip(T node);
   string GetImageUrl(string operation);
}
```

Výpis 2: Rozhraní ITreeCreator

Seznam tříd implementujících toto rozhraní:

- `MsSqlTree` – třída pro generování stromu z plánu vykonání dotazu databáze MS SQL Server
- `OracleTree` – třída pro generování stromu z plánu vykonání dotazu databáze Oracle
- `MySqlTree` – třída pro generování stromu z plánu vykonání dotazu databáze MySQL
- `PostgreSqlTree` – třída pro generování stromu z plánu vykonání dotazu databáze PostgreSQL

Pro samotnou vizualizaci plánů vykonání dotazů je použita standardní komponenta ASP.NET WebForms, TreeView.

5.2 Schéma databázového modelu

5.2.1 Seznam relací

- `Customer` (CustomerID, FirstName, LastName, Title, Street, City, Region, PostalCode, Country, Phone)
- `OrderOfProduct` (OrderOfProductID, OrderDate, ShippedDate, ShipAddress, Freight, CustomerID, EmployeeID)
- `OrderDetail` (UnitPrice, Quantity, Discount, ProductID, OrderOfProductID)
- `Product` (ProductID, ProductName, UnitPrice, UnitsInStock, UnitsOnOrder, Discontinued, CategoryID)
- `Category` (CategoryID, CategoryName, Description)
- `Employee` (EmployeeID, FirstName, LastName, Title, Street, City, Region, PostalCode, Country, Phone, BirthDate, HireDate, Notes)

Relace `Customer`(zákazník) obsahuje informace o zákaznících, jejich křestní jméno, příjmení, titul, informace o bydlišti a telefonní kontakt. Primární klíč relace je CustomerID.

Relace `OrderOfProduct`(objednávka) obsahuje informace o objednávkách zboží, o datu objednání, datu doručení, adrese doručení a o ceně za dopravu. Primární klíč relace je OrderOfProductID a vedlejší klíče jsou CustomerID, který identifikuje zákazníka, který zboží nakoupil, a EmployeeID, který označuje zaměstnance, který zboží prodal.

Relace `OrderDetail`(detail objednávky) obsahuje podrobné informace o konkrétní objednávce zboží, informace o ceně za jednotku, množství a případné slevě. Cizí klíče jsou ProductID, který označuje nakoupené zboží, a OrderOfProductID, který označuje objednávku, ke které patří tyto popisující informace. Cizí klíče tvoří dohromady složený primární klíč.

Relace `Product` obsahuje informace o jednotlivých produktech, jejich název, cenu za jednotku, množství na skladě, množství objednaného zboží, příznak zdali je u tohoto produktu přerušen prodej (0 – není přerušen, 1 – je přerušen). Primární klíč relace je ProductID a vedlejší klíč je CategoryID, který určuje kategorii daného zboží.

Relace `Category`(kategorie) obsahuje informace o kategoriích produktů, které do těchto kategorií spadají. Obsahuje název kategorie a popis kategorie. Primárním klíčem je CategoryID.

Relace `Employee`(zaměstnanec) obsahuje informace o zaměstnancích, jejich křestní jméno, příjmení, titul, informace o bydlišti, telefonní kontakt, datum narození, datum navázání a ukončení pracovního poměru a případné další poznámky k zaměstnanci. Primární klíč relace je EmployeeID.

Při návrhu tohoto schématu jsem se řídil zásadami pro vytvoření kvalitního databázového schématu a také jsem se snažil vytvořit takové schéma, na kterém se dá simulovat co nejvíce typů operací při provádění SQL dotazů. Do databáze byly vloženy vygenerovaná data v takovém

Tabulka 5: Seznam tabulek s počtem vložených záznamů

Tabulka	**Počet záznamů**
Category	500
Product	100 000
Employee	20 000
Customer	200 000
OrderOfProduct	800 000
OrderDetail	2 045 000

množství, abychom se při simulaci provádění dotazů přiblížili reálným scénářům, které mohou v praxi nastat. Počet záznamů v jednotlivých tabulkách je znázorněn v tabulce 5. Schéma databáze v příloze A.

5.2.2 Fyzická implementace schématu

Jelikož se fyzické implementace v jednotlivých SŘBD liší, je potřeba si specifikovat, jak bylo schéma v jednotlivých databázích vytvořeno.

- MS SQL Server – tabulka Category byla vytvořena jako tabulka typu halda. Ostatní byly vytvořeny jako klastrovaný index.
- Oracle – tabulka Category byla vytvořena jako tabulka typu halda. Ostatní byly vytvořeny jako IOT.
- MySQL – všechny tabulky byly vytvořeny jako klastrovaný index.
- PostgreSQL – všechny tabulky jsou tabulky typu halda.

5.3 Ukázky vizualizovaných plánů vykonání dotazu

Pro ukázku vizualizace plánů vykonání dotazů jsem použil tři různé SQL dotazy, u kterých je dobře viditelné, jak se provádění SQL dotazu v jednotlivých SŘBD liší. Také je v těchto ukázkách možné vypozorovat, jak podrobné plány vykonání dotazu v jednotlivých SŘBD jsou. V tabulce 6 jsou zobrazeny rychlosti provedení jednotlivých SQL dotazů.

Tabulka 6: Rychlosti provedení SQL dotazů [s]

	MS SQL Server	Oracle	MySQL	PostgreSQL
Ukázka č.1	9,03	1,23	46,40	1,56
Ukázka č.2	0,52	0,17	326,64	0,08
Ukázka č.3	2,25	8,33	20,66	4,66

5.3.1 Ukázka č.1

```
SELECT DISTINCT c.CustomerID, c.FirstName, c.LastName FROM Customer c
JOIN OrderOfProduct op ON op.CustomerID = c.CustomerID
JOIN OrderDetail od ON od.OrderOfProductID = op.OrderOfProductID
WHERE od.Discount >= 0.9;
```

Výpis 3: SQL dotaz z ukázky č.1

Prvním dotazem bychom chtěli zjistit zákazníky, kteří někdy provedli nákup zboží se slevou 90% a vyšší.

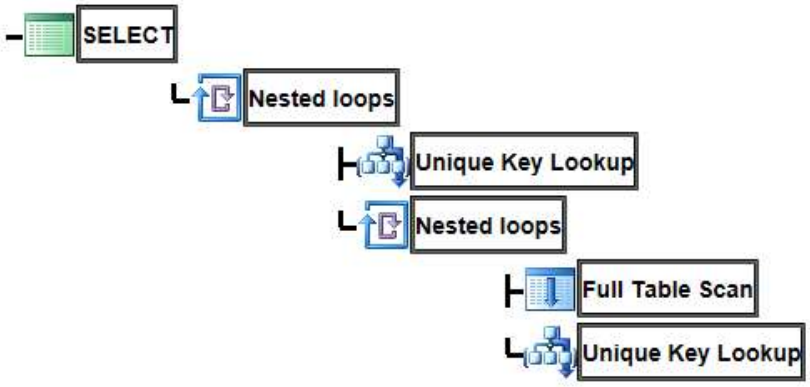
- **MS SQL Server**

  Provedlo se čtení klastrovaného indexu OrderDetail a následně hašované spojení s výstupem čtení neklastrovaného indexu CustomerID tabulky OrderOfProduct. Následně proběhlo čtení klastrovaného indexu Customer a spojení s předchozí relací. V plánu vykonání dotazu je také vidět, že dotaz byl vykonán paralelně. Plán vykonání dotazu je znázorněn v příloze B, na obrázku 16.

- **Oracle**

  Bylo provedeno čtení IOT Customer a následné spojení vnořenými cykly s výstupem čtení indexu CustomerID tabulky OrderOfProduct. Následně proběhl průchod indexem CustomerID tabulky OrderOfProduct a hašované spojení s předchozí relací. Následně proběhlo čtení IOT OrderDetail a spojení s výstupem předchozí relace. Následně se pomocí operace Hash-Unique eliminovaly duplicity. Plán vykonání dotazu je znázorněn v příloze B, na obrázku 17.

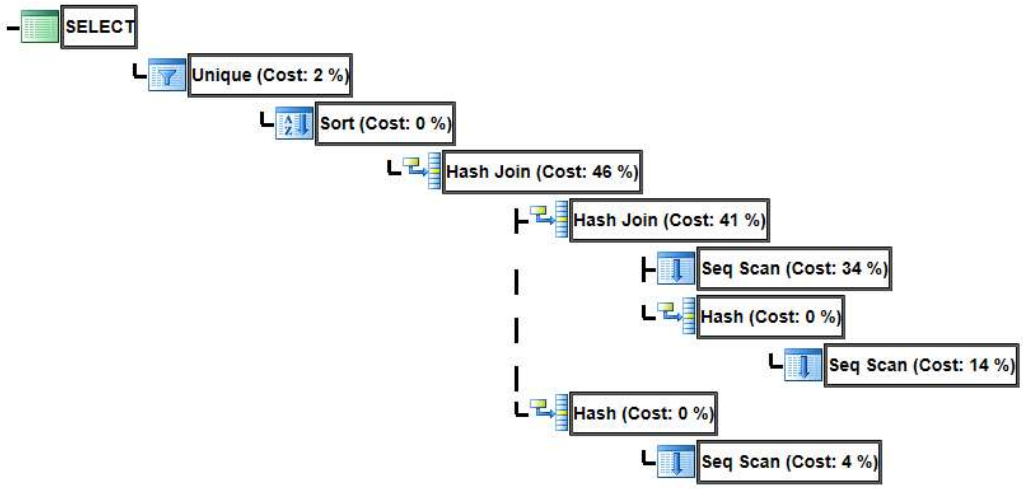
- **MySQL**



Obrázek 6: Plán vykonání dotazu z ukázky č.1 v MySQL

Došlo k průchodu klastrovaného indexu OrderDetail a spojení vnořenými cykly s výstupem čtení indexu OrderOfProduct. Následně proběhlo čtení indexu Customer a spojení vnořenými cykly s předchozí relací. Plán vykonání dotazu je znázorněn na obrázku 6.

- **PostgreSQL**



Obrázek 7: Plán vykonání dotazu z ukázky č.1 v PostgreSQL

Došlo tedy k sekvenčnímu průchodu tabulky OrderOfProduct a hašovanému spojení s výstupem sekvenčního průchodu tabulky OrderDetail. Následně došlo k sekvenčnímu čtení tabulky Customer a hašovanému spojení s výstupem předchozí relace. Unikátnosti výsledku bylo zajištěno setříděním výstupu a následnou operací Unique. Plán vykonání dotazu je znázorněn na obrázku 7.

5.3.2 Ukázka č.2

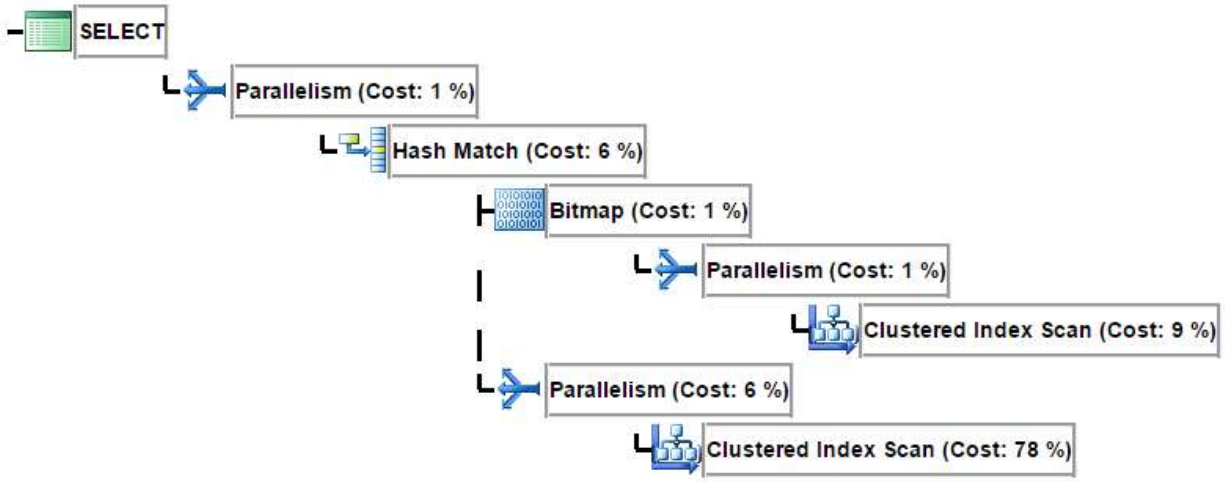
```
SELECT c.FirstName, c.LastName, c. Street
FROM Customer c WHERE c.Street IN
(
   SELECT e.Street FROM Employee e
);
```

Výpis 4: SQL dotaz z ukázky č.2

Druhým dotazem bychom chtěli zjisti zákazníky, kteří bydlí ve stejné ulici, jako někdo ze zaměstnanců.

- **MS SQL Server**

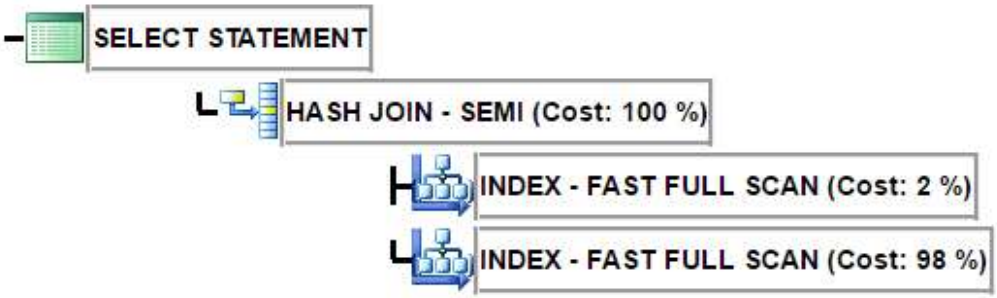
  Plán vykonání dotazu je velmi jednoduchý, provedlo se čtení klastrovaných indexů Employee a Customer, jejíž výstupy byly spojeny pomocí hašovaného spojení. Opět byl dotaz proveden paralelně. Plán vykonání dotazu je znázorněn na obrázku 8.



Obrázek 8: Plán vykonání dotazu z ukázky č.2 v MS SQL

- **Oracle**

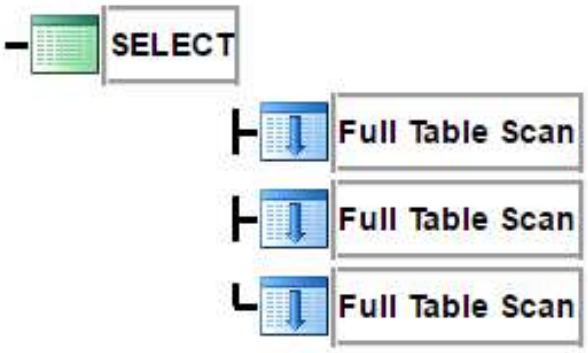
  Bylo provedeno čtení IOT Employee a Customer s následným spojením pomocí hašovaného spojení. Plán vykonání dotazu je znázorněn na obrázku 9.



Obrázek 9: Plán vykonání dotazu z ukázky č.2 v Oracle

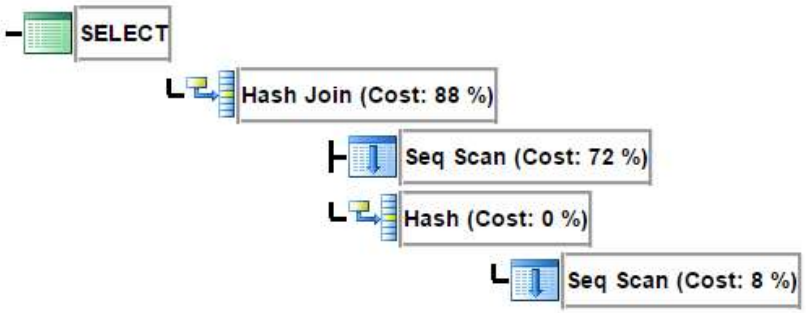
- **MySQL**

  Z plánu vykonání dotazu není bohužel možné vyčíst všechny podrobnosti, které bychom o způsobu vykonání dotazu chtěli vědět, ale hned na první pohled je vidět, že oproti ostatním SŘBD bylo provedeno čtení navíc. Čtení z klastrovaného indexu Employee je vloženo do pod-dotazu, jehož výstup se následně uložil do dočasné tabulky. Tento fakt je znázorněn v atributu Select-type: Materialized. Proběhl tedy průchod klastrovaného indexu Employee, následné uložení do dočasné tabulky, průchod klastrovaného indexu Customer a následné spojení, které však také na plánu vykonání dotazu není vidět, je však naznačeno v atributu Extra: Block Nested Loop. Plán vykonání dotazu je znázorněn na obrázku 10.



Obrázek 10: Plán vykonání dotazu z ukázky č.2 v MySQL

- **PostgreSQL**



Obrázek 11: Plán vykonání dotazu z ukázky č.2 v PostgreSQL

Bylo provedeno čtení tabulek Emloyee a Customer a následné hašované spojení výstupů těchto dvou relací. Plán vykonání dotazu je znázorněn na obrázku 11.

5.3.3 Ukázka č.3

```
SELECT c.CustomerID, c.FirstName, c.LastName, op.OrderDate FROM Customer c
JOIN OrderOfProduct op ON op.CustomerID = c.CustomerID
GROUP BY c.CustomerID, c.FirstName, c.LastName, op.OrderDate
HAVING op.OrderDate >= ALL
(
   SELECT op1.OrderDate FROM OrderOfProduct op1
   WHERE op1.CustomerID = c.CustomerID
);
```

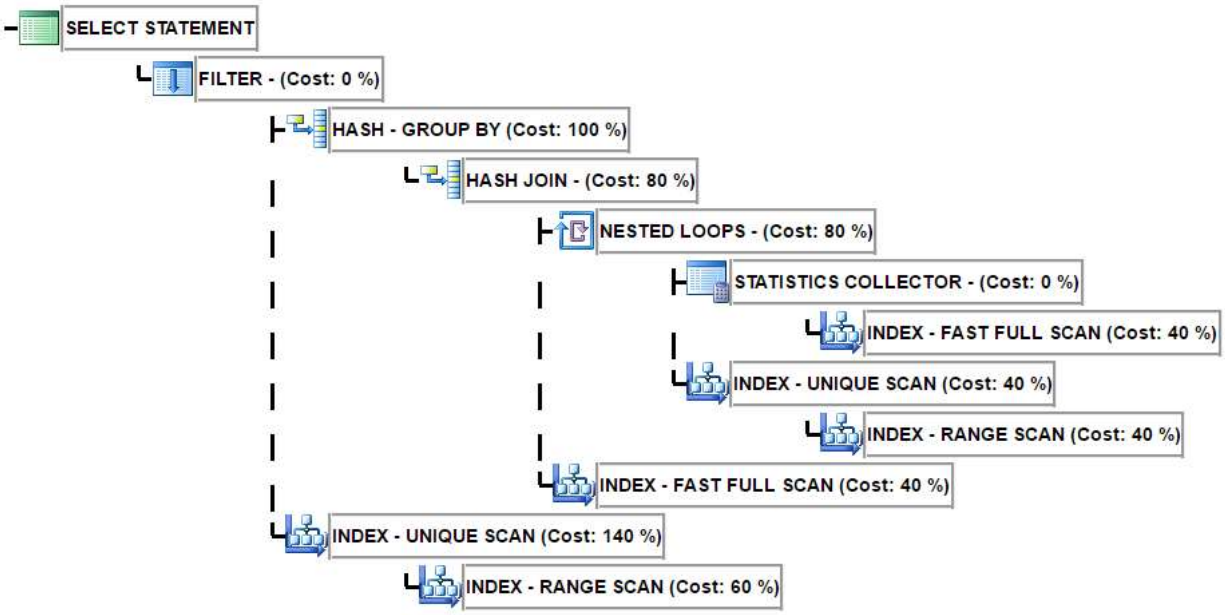
Výpis 5: SQL dotaz z ukázky č.3

Třetím dotazem bychom chtěli vypsat všechny zákazníky a data jejich poslední objednávky.

- **MS SQL Server**

  Bylo provedeno čtení klastrovaného indexu Customer a OrderOfProduct a jejich následné spojení pomocí hašovaného spojení. Následně došlo ke groupnutí přes požadované sloupce a další čtení klastrovaného indexu OrderOfProduct z vnořeného dotazu. Následně došlo opět k hašovanému spojení. Dotaz byl opět zpracován paralelně. Plán vykonání dotazu je znázorněn v příloze B, na obrázku 18.

- **Oracle**



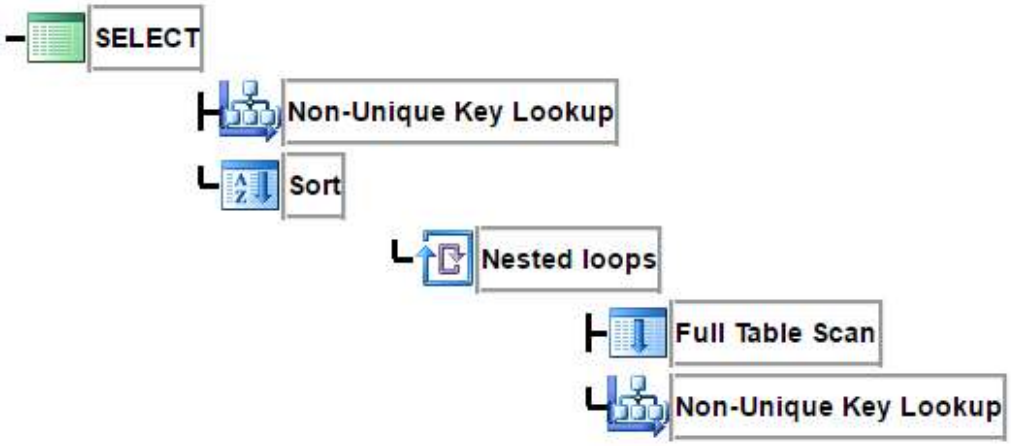
Obrázek 12: Plán vykonání dotazu z ukázky č.3 v Oracle

Proběhl průchod indexu CustomerID tabulky OrderOfProduct následovaný čtením IOT OrderOfProduct a následně spojení vnořenými cykly s výstupem čtení IOT Customer. Následně proběhlo čtení IOT OrderOfProduct a hašované spojení s předchozí relací. Následně došlo k provedení operace grupování předchozí relace. Dále proběhl průchod indexu CustomerID tabulky OrderOfProduct následovaný čtením IOT OrderOfProduct a následná filtrace záznamů. Plán vykonání dotazu je znázorněn na obrázku 12.

- **MySQL**

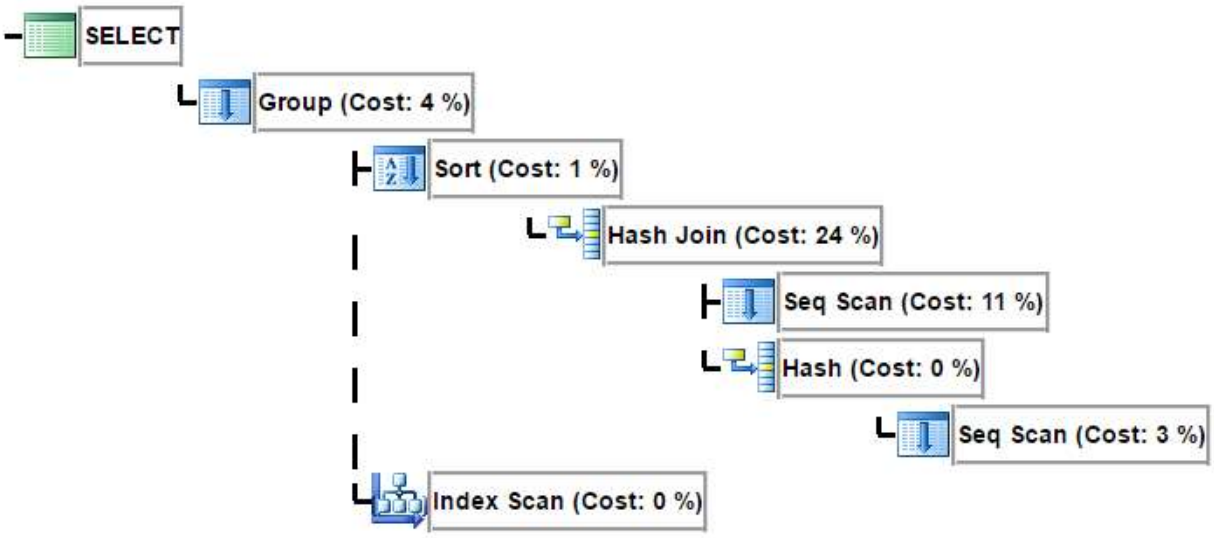
  MySQL provedl čtení z neunikátního indexu CustomerID tabulky OrderOfProduct a následné spojení s výsledkem průchodu klastrovaného indexu Customer. Zde je možné vidět, že MySQL nijak neznázorňuje přístup do klastrovaného indexu pomocí klíče indexu. Při průchodu neunikátním indexem CustomerID tabulky OrderOfProduct totiž není naznačeno získání OrderDate. Následně proběhlo spojení vnořenými cykly, setřídění a následné čtení neunikátního indexu CustomerID tabulky OrderOfProduct. Opět se pravděpodobně

museloo přistupovat k datům v klastrovaném indexu, což není v plánu vykonání dotazu naznačeno. Plán vykonání dotazu je znázorněn na obrázku 13.



Obrázek 13: Plán vykonání dotazu z ukázky č.3 v MySQL

- **PostgreSQL**



Obrázek 14: Plán vykonání dotazu z ukázky č.3 v PostgreSQL

Došlo ke čtení tabulek Customer a OrderOfProduct a následnému hašovanému spojení těchto dvou relací. Dále byl výstup setříděn a došlo ke čtení indexu CustomerID tabulky OrderOfProduct a groupnutí přes požadované sloupce. Plán vykonání dotazu je znázorněn na obrázku 14.

5.4 Zhodnocení vizualizace

Na jednotlivých ukázkách jsme si představili vizualizované plány vykonání dotazů ve vybraných SŘBD. Na ukázkách je vidět, že množství informací o provedení SQL dotazů se v jednotlivých SŘBD liší, avšak ty nejdůležitější informace jsou téměř vždy poskytnuty. Tabulka 7 srovnává množství podrobností o plánech vykonání dotazů v jednotlivých SŘBD.

Tabulka 7: Srovnání plánů vykonání dotazů

	MS SQL Server	**Oracle**	**MySQL**	**PostgreSQL**
Název operace	✓	✓	✗	✓
Počet řádků na vstupu	✓	✓	✓	✓
Cena operace	✓	✗	✗	✗
Cena operace včetně potomků	✓	✓	✗	✓
Rozlišení ceny pro CPU a I/O	✓	✓	✗	✗
Průměrná velikost řádku	✓	✗	✗	✓
Seznam sloupců na výstupu	✓	✓	✗	✓
Název použitého objektu	✓	✓	✗	✗
Alias pro použitý objekt	✓	✓	✓	✓
Podmínka spojení	✓	✓	✓	✓
Podmínka filtrace	✓	✓	✓	✓

Na ukázce č.2 v MySQL je vidět, že tento SŘBD nenabízí veškeré potřebné informace pro korektní vizualizaci plánu vykonání dotazu. U ostatních SŘBD je také vidět, že mezi jednotlivými kroky plánu vykonání dotazu se vyskytují i operace, které nemají žádné zastoupení mezi logickými ani fyzickými operacemi a jsou do plánů přidány jen jako přidaná informace o průběhu vykonání SQL dotazů. Takovou operací může být například Statistics Collector v Oraclu. Některé SŘBD naopak určité operace naopak skrývají, jako MySQL operaci pro eliminaci duplicitních záznamů, případně operace spojení.

Kromě samotných informací o plánech vykonání dotazů je potřeba hodnotit i formáty, ve kterých se dají z SŘBD plány vykonání dotazů získat. Při tvorbě aplikace jsem volil ty formáty, které se daly nejlépe zpracovat a zobrazit v aplikaci. MS SQL Server a Oracle ve formátu tabulky nabízejí všechny potřebné informace pro vytvoření stromové struktury plánu vykonání dotazu. PostgreSQL ve formátu XML nabízí také velmi slušné možnosti pro další zpracování do stromové struktury, avšak je potřeba si nejprve jednotlivé uzly XML označit identifikátorem a to vede k prodloužení doby implementace. MySQL je na tom celkově nejhůře. Máme možnost si vybrat výpis do tabulky, ze kterého získáme jen malé množství informací, anebo formát JSON, který sice obsahuje informací více, ale díky své nekonzistentní struktuře se velmi špatně zpracovává.

6 Závěr

V dnešní době je už velmi dobře znám fakt, že z plánů vykonání dotazů se dají zjistit velmi užitečné informace pro optimalizaci SQL dotazů, a podle toho je brán důraz na jejich zobrazování. Všechny běžně používané SŘBD nabízejí zobrazování plánů vykonání dotazů ať už v různých textových podobách, rozšířených formátech jako XML nebo JSON, případně graficky zobrazené.

V bakalářské práci jsme si popsali, jak jsou plány vykonání dotazů užitečné při optimalizaci a psaní SQL dotazů. Dále jsme porovnali plány vykonání dotazů v dnešních SŘBD a možnosti pro jejich získání. Cíl bakalářské práce, vytvořit systém pro vizualizaci plánů vykonání dotazů, byl splněn. Na tomto systému jsme si mohli ověřit, jaké možnosti vizualizace plánů vykonání dotazů jednotlivé SŘBD nabízejí. Do budoucna je možné systém rozšířit o možnost provádění importu dat pomocí bulk operací, také o možnost komunikovat s vícero SŘBD a změnit formát zpracování v MySQL na JSON, kvůli získání podobnějších informací o plánech vykonání dotazů.

Tato práce byla pro mě velkým přínosem. Rozšířil jsem si znalosti o jednotlivých SŘBD, prohloubil jsem své dosavadní schopnosti programování v jazyce C# a také jsem získal cenné zkušenosti se studiem odborné literatury. Tato práce mě utvrdila v tom, že databázové systémy jsou velmi různorodé a nabízejí komplexní řešení ukládání dat.

Literatura

[1] FRITCHEY, Grant. *The art of high performance SQL code: SQL server execution plans.* Cambridge, UK: Simple-Talk Pub, 2008. ISBN 9781906434021.

[2] SCHWARTZ, Baron. a Jeremy D. ZAWODNY. *High performance MySQL.* 2nd ed. Sebastopol: O'Reilly, c2008. ISBN 978-0596101718.

[3] NEVAREZ, Benjamin. *Inside the SQL Server Query Optimizer.* S.l.: Red Gate Books, 2011. ISBN 9781906434601.

[4] LIGHTSTONE, Sam., Toby J. TEOREY a Tom NADEAU. *Physical database design: the database professional's guide to exploiting indexes, views, storage, and more.* Boston: Morgan Kaufmann/Elsevier, c2007. ISBN 978-0123693891.

[5] SHASHA, Dennis Elliott. a Philippe BONNET. *Database tuning: principles, experiments, and troubleshooting techniques.* [Rev. ed.]. San Francisco, CA: Morgan Kaufmann Publishers, c2003. ISBN 978-1558607538.

[6] SMITH, Gregory. *PostgreSQL 9.0: high performance : accelerate your PostgreSQL system and avoid the common pitfalls that can slow it down.* Birmingham: Packt Publishing, c2010. ISBN 9781849510301.

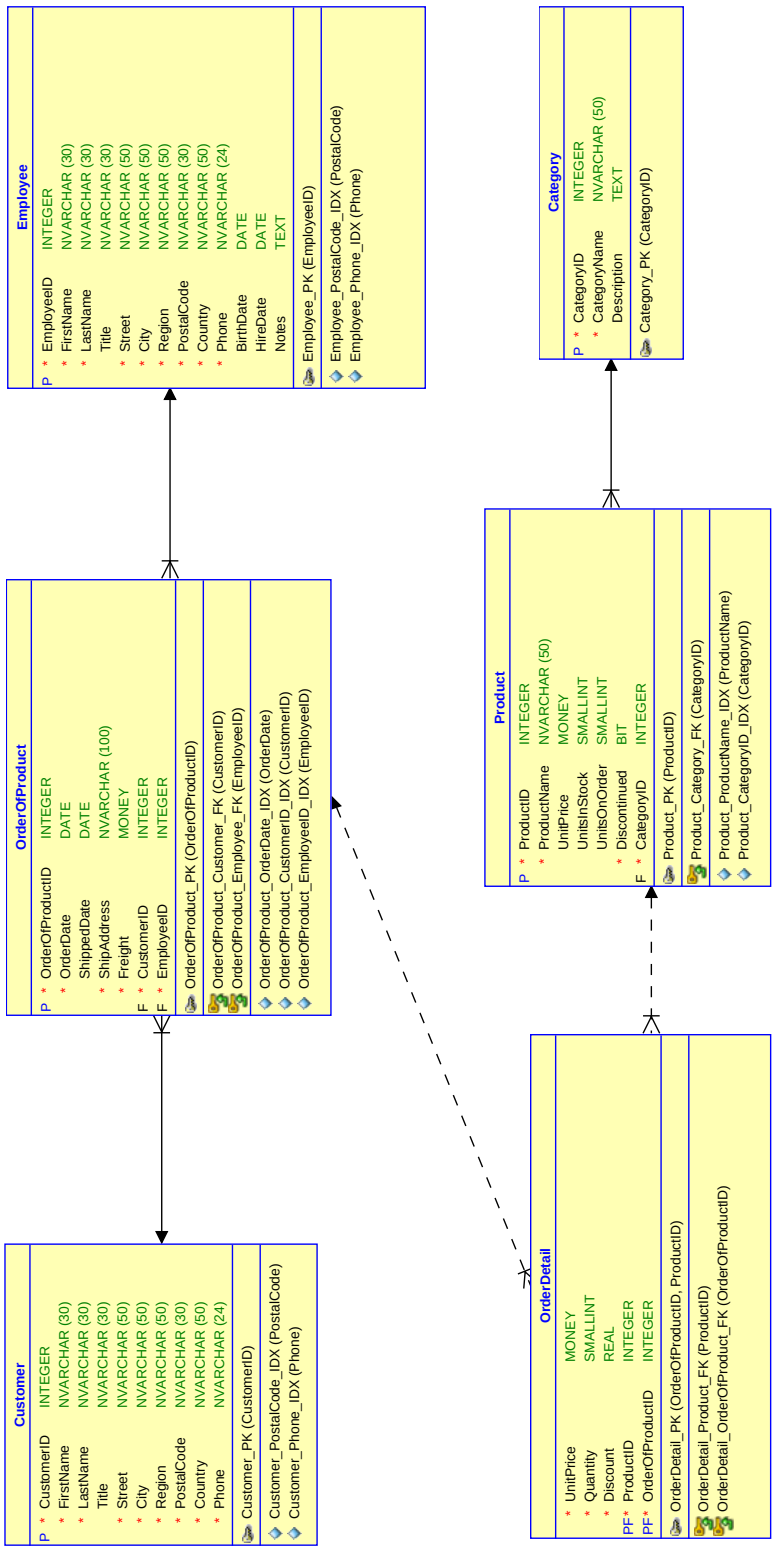
[7] Oracle. *Using EXPLAIN PLAN | Oracle Database Online Documentation, 10g Release 2* [online]. Copyright © 2017 Oracle Corporation [cit. 15.04.2017]. Dostupné z: `https://docs.oracle.com/cd/B19306_01/server.102/b14211/ex_plan.htm#i18300`.

[8] Oracle. *The Query Optimizer | Oracle Database Online Documentation, 10g Release 2* [online]. Copyright © 2017 Oracle Corporation [cit. 15.04.2017]. Dostupné z: `https://docs.oracle.com/cd/B19306_01/server.102/b14211/optimops.htm#i44963`.

[9] MySQL. *EXPLAIN Output Format | MySQL 5.7 Reference Manual* [online]. Copyright © 2017 Oracle Corporation [cit. 15.04.2017]. Dostupné z: `https://dev.mysql.com/doc/refman/5.7/en/explain-output.html`.

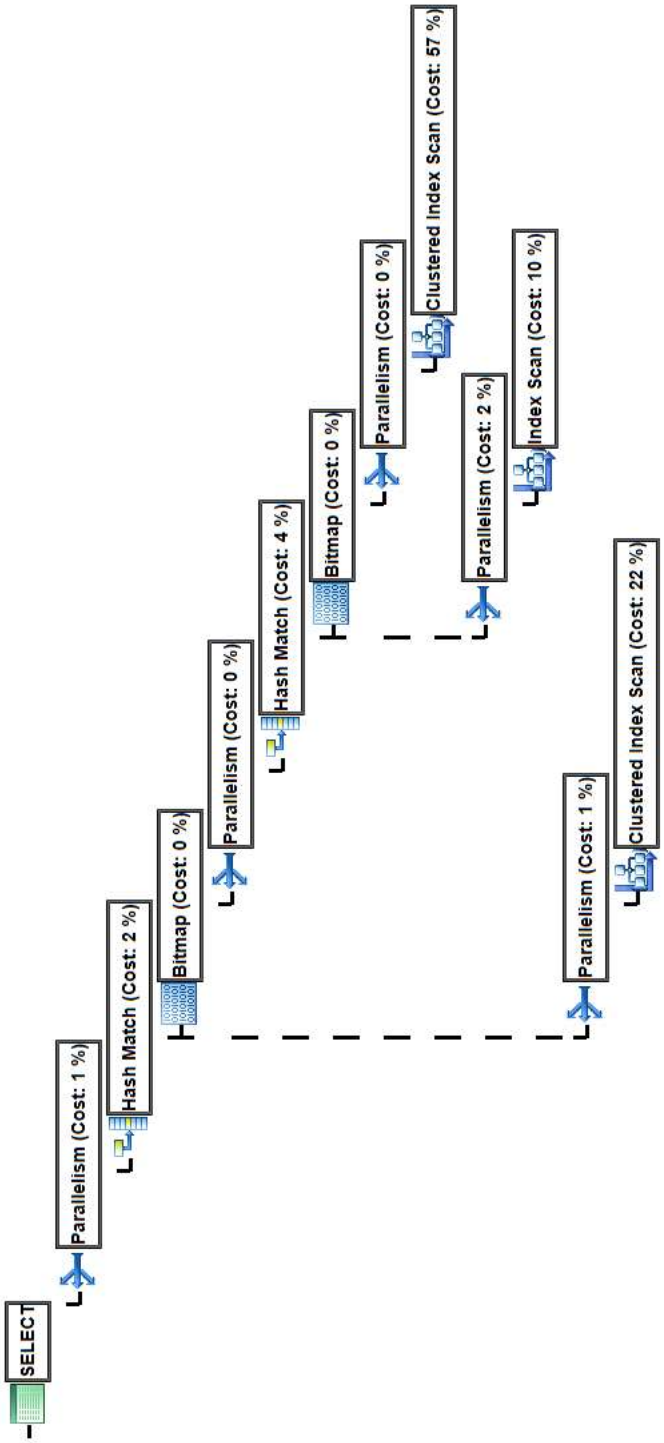
[10] Microsoft. *SET SHOWPLAN ALL (Transact-SQL) | Microsoft Docs* [online]. Dostupné z: `https://docs.microsoft.com/en-us/sql/t-sql/statements/set-showplan-all-transact-sql`.

A Schéma databáze

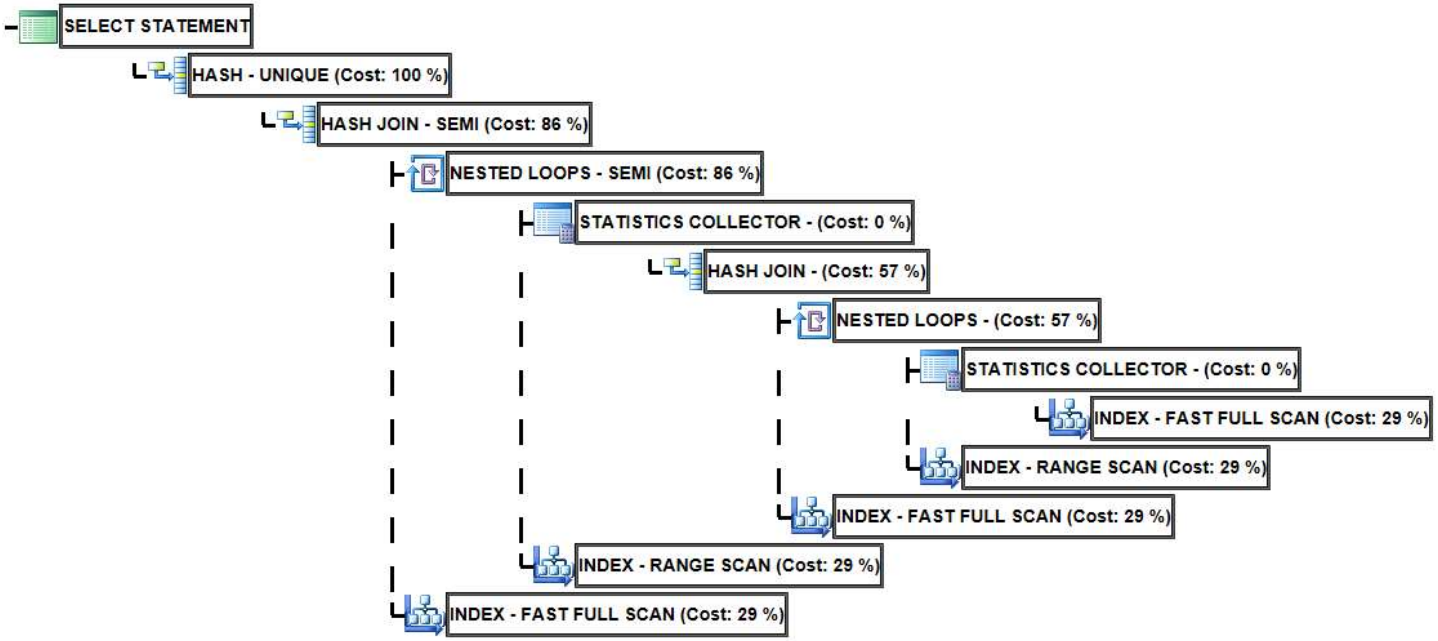


Obrázek 15: Schéma databáze

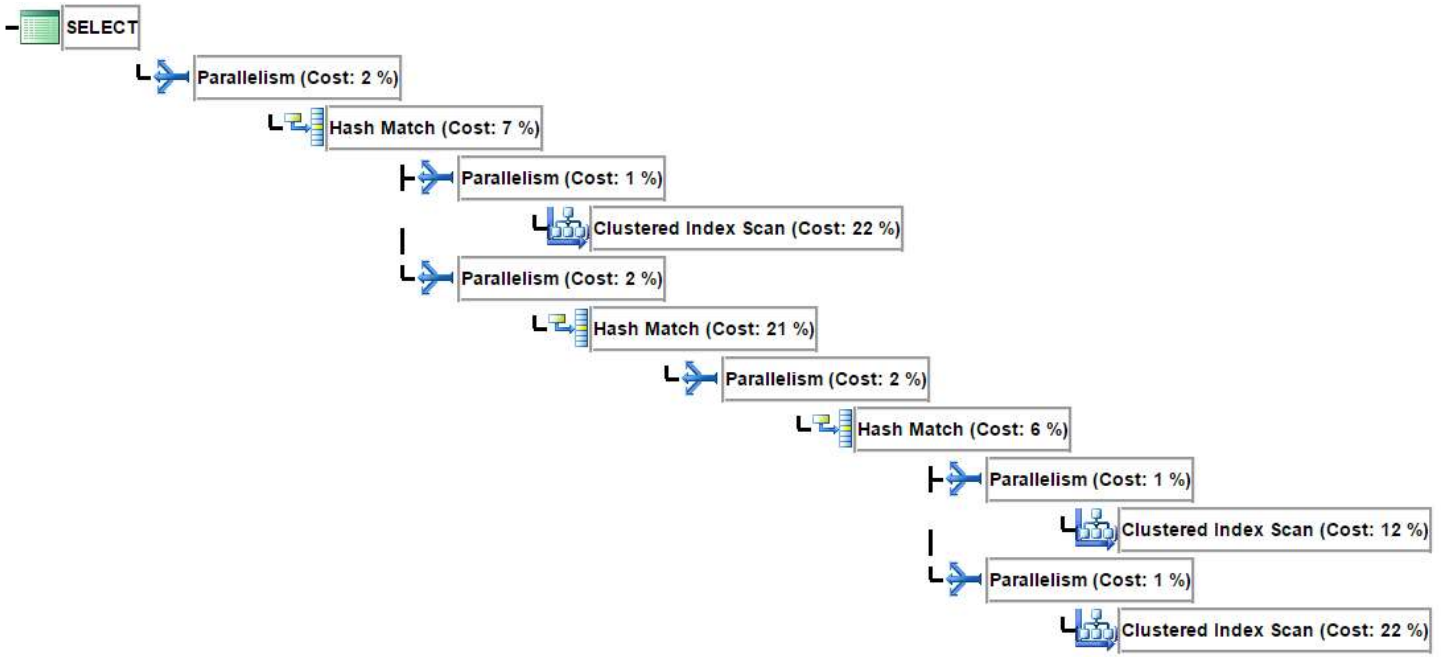
B Vizualizované plány vykonání dotazů



Obrázek 16: Plán vykonání dotazu z ukázky č.1 v MS SQL



Obrázek 17: Plán vykonání dotazu z ukázky č.1 v Oracle



Obrázek 18: Plán vykonání dotazu z ukázky č.3 v MS SQL

C Obsah DVD

Na přiloženém DVD jsou uložená následující data:

- Text bakalářské práce ve formátu pdf
- SQL skripty pro vytvoření databází
- Zdrojové kódy systému QEP